Moab Workload Manager

Administrator Guide 10.1.0

March 2025

Adaptive
COMPUTING

Legal Notices

Adaptive Computing Enterprises, Inc.
1100 5th Ave S, Suite #201
Naples, FL 34102
+1 (239) 330-6093
www.adaptivecomputing.com

Contents

Moab Workload Manager Overview

Welcome to the *Moab Workload Manager Administrator Guide 10.1.0*

This guide is intended for Moab Workload Manager system administrators.

Moab Workload Manager (a.k.a. Moab) is a scheduling and management system designed for clusters, grids, and on-demand/utility computing systems. Moab:

- Applies site policies and extensive optimizations to orchestrate jobs, services, and other workload across the ideal combination of network, compute, and storage resources.
- Enables Adaptive Computing; allowing compute resources to be customized to changing needs and failed systems to be automatically fixed or replaced.
- Increases system resource availability, offers extensive cluster diagnostics, delivers powerful quality of service (QoS) and service level agreement (SLA) features, and it provides rich visualization of cluster performance through advanced statistics, reports, and charts. In addition, the Elastic Computing feature enables Moab to temporarily utilize systems that can provide additional resources to take care of increased workload demand (caused by high job backlog) in a more timely manner.

Moab also works with major resource management and resource monitoring tools. From hardware monitoring systems such as IPMI to provisioning systems and storage managers, Moab takes advantage of domain expertise to allow these systems to do what they do best, importing their state information and providing them with the information necessary to do their job better. Moab uses its global information to coordinate the activities of both resources and services, which optimizes overall performance in-line with high-level mission objectives.

Chapter 1: Philosophy and Goals of Moab Workload Manager

The scheduler's purpose is to optimally use resources in a convenient and manageable way. System users want to specify resources, obtain quick turnaround on their jobs, and have reliable resource allocation. On the other hand, admins want to understand both the workload and the resources available. This includes current state, problems, and statistics—information about what is happening that is transparent to the end user. Admins need an extensive set of options to enable management enforced policies and tune the system to obtain desired statistics.

There are other systems that provide batch management; however, Moab is unique in many respects. Moab matches jobs to nodes, dynamically reprovisions nodes to satisfy workload, and dynamically modifies workload to better take advantage of available nodes. Moab enables sites to fully visualize cluster and user behavior. It can integrate and orchestrate resource monitors, databases, identity managers, license managers, networks, and storage systems, therefore providing a cohesive view of the cluster—a cluster that fully acts and responds according to site mission objectives.

Moab can dynamically adjust security to meet specific job needs. Moab can create real and virtual clusters on demand and from scratch that are custom-tailored to a specific request. Moab can integrate visualization services, web farms, and application servers; it can also create powerful grids of disparate clusters. Moab maintains complete accounting and auditing records, exporting this data to information services on command, even providing professional billing statements to cover all used resources and services.

Moab provides user-centric and application-centric web portals and powerful graphical tools for monitoring and controlling every conceivable aspect of a cluster's objectives, performance, workload, and usage. Moab is unique in its ability to deliver a powerful user-centric cluster with little effort. Its design is focused on ROI, better use of resources, increased user effectiveness, and reduced staffing requirements.

In this chapter:

1.1 Value of a Batch System

Batch systems provide centralized access to distributed resources through mechanisms for submitting, launching, and tracking jobs on a shared resource. This greatly simplifies use of the cluster's distributed resources, allowing users a *single system image* in terms of managing jobs and aggregate compute resources available. Batch systems should do much more than just provide a global view of the cluster, though. Using compute resources in a fair and effective manner is complex, so a scheduler is necessary to determine when, where, and how to run jobs to optimize the cluster. This section describes the categories of scheduling decisions.

Traffic Control

A scheduler must prevent jobs from interfering. If jobs contend for resources, cluster performance decreases, job execution is delayed, and jobs may fail. Therefore, the scheduler tracks resources and dedicates requested resources to a particular job, which prevents use of such resources by other jobs.

Mission Policies

Clusters and other HPC platforms typically have specific purposes; to fulfill these purposes, or mission goals, there are usually rules about system use pertaining to who or what is allowed to use the system. To be effective, a scheduler must provide a suite of policies allowing you to *map* site mission policies into scheduling behavior.

Optimizations

The compute power of a cluster is a limited resource; over time, demand inevitably exceeds supply. Intelligent scheduling decisions facilitate higher job volume and faster job completion. Though subject to the constraints of the traffic control and mission policies, the scheduler must use whatever freedom is available to maximize cluster performance.

1.2 Philosophy and Goals

Managers want high system utilization and the ability to deliver various qualities of service to various users and groups. They need to understand how available resources are delivered to users over time. They also need admins to tune *cycle delivery* to satisfy the current site mission objectives.

Determining a scheduler's success is contingent upon establishing metrics and a means to measure them. The value of statistics is best understood if optimal statistical values are known for a given environment, including workload, resources, and policies. That is, if an admin could determine that a site's typical workload obtained an average queue time of 3.0

hours on a particular system, that would be a useful *statistic*; however, if an admin knew that through proper tuning the system could deliver an average queue time of 1.2 hours with minimal negative side effects, that would be valuable *knowledge*.

Moab development relies on extensive feedback from users, admins, and managers. At its core, it is a tool designed to *manage* resources and provide meaningful information about what is actually happening on the system.

Management Goals

A manager must ensure that a cluster fulfills the purpose for which it was purchased, so a manager must deliver cycles to those projects that are most critical to the success of the funding organizations. Management tasks to fulfill this role may include the following:

- Define cluster mission objectives and performance criteria
- Evaluate current and historical cluster performance
- Instantly graph delivered service

Administration Goals

An admin must ensure that a cluster is effectively functioning within the bounds of the established mission goals. Admins translate goals into cluster policies, identify and correct cluster failures, and train users in best practices. Given these objectives, an admin may be tasked with each of the following:

- Maximize utilization and cluster responsiveness
- Tune fairness policies and workload distribution
- Automate time-consuming tasks
- Troubleshoot job and resource failures
- Instruct users of available policies and in their use regarding the cluster
- Integrate new hardware and cluster services into the batch system

End User Goals

End users are responsible for learning about the resources available, the requirements of their workload, and the policies to which they are subject. Using this understanding and the available tools, they find ways to obtain the best possible responsiveness for their own jobs. A typical end user may have the following tasks:

- Manage current workload
- Identify available resources
- Minimize workload response time

- Track historical usage
- Identify effectiveness of prior submissions

1.3 Workload

Moab can manage a broad spectrum of compute workload types, and it can optimize all workload types within the same cluster simultaneously, delivering on the objectives most important to each workload type, as described in this topic.

Batch Workload

Batch workload is characterized by a *job* command file that typically describes all critical aspects of the needed compute resources and execution environment. With a batch job, the job is submitted to a job queue and runs somewhere on the cluster as resources become available. In most cases, the submitter submits multiple batch jobs with no execution time constraints and processes job results as they become available.

Moab can enforce rich policies defining how, when, and where batch jobs run to deliver compute resources to the most important workload and provide general SLA guarantees while maximizing system utilization and minimizing average response time.

Interactive Workload

Interactive workload differs from batch in that requestors are interested in immediate response and are generally waiting for the interactive request to be executed before going on to other activities. In many cases, interactive submitters will continue to be *attached* to the interactive job, routing keystrokes and other input into the job and seeing both output and error information in real-time. While interactive workload can be submitted within a job file, commonly, it is routed into the cluster via a web or other graphical terminal and the end user may never even be aware of the underlying use of the batch system.

For managing interactive jobs, the focus is usually on setting aside resources to guarantee immediate execution or at least a minimal wait time for interactive jobs. Targeted service levels require management when mixing batch and interactive jobs. Interactive and other job types can be dynamically steered in terms of what they are executing, and in terms of the quantity of resources required by the application.

Calendar Workload

Calendar workload must be executed at a particular time and possibly in a regular periodic manner. For such jobs, time constraints range from flexible to rigid. For example, some calendar jobs may need to complete by a certain time, while others must run exactly at a given time each day or each week.

Moab can schedule the future and can therefore guarantee resource availability at needed times to allow calendar jobs to run as required. Furthermore, Moab provisioning features can locate or temporarily create the needed compute environment to properly execute the target applications.

Service Workload

Moab can schedule and manage both individual applications and long-running or persistent services. Service workload processes externally-generated transaction requests while Moab provides the distributed service with needed resources to meet target backlog or response targets to the service. Examples of service workload include parallel databases, web farms, and visualization services. Moab can apply cluster, grid, or dynamically-generated on-demand resources to the service.

When handling service workload, Moab observes the application in a highly abstract manner. Using the JOBCFG parameter, aspects of the service jobs can be discovered or configured with attributes describing them as resource consumers possessing response time, backlog, state metrics, and associated QoS targets. In addition, each application can specify the type of compute resource required (OS, arch, memory, disk, network adapter, data store, and so forth), and the support environment (network, storage, external services, and so forth).

If the QoS response time/backlog targets of the application are not being satisfied by the current resource allocation, Moab evaluates the needs of this application against all other site mission objectives and workload needs and determines what it must do to locate or create (that is, provision, customize, secure) the needed resources. With the application resource requirement specification, you can also indicate proximity/locality constraints, partition policies, ramp-up/ramp-down rules, and so forth.

Once Moab identifies and creates appropriate resources, it hands these resources to the application via a site customized URL. This URL can be responsible for whatever application-specific handshaking must be done to launch and initialize the needed components of the distributed application upon the new resources. Moab engages in the hand-off by providing needed context and resource information and by launching the URL at the appropriate time.

Related Topics

- 18.3 Malleable Jobs
- 6.3 Quality of Service (QoS) Facilities

Chapter 2: Scheduler Basics

In this chapter:

2.1 Initial Moab Configuration

In this section:

2.1.1 moab.cfg

After Moab is installed, there may be minor configuration remaining within the primary configuration file, `moab.cfg`. While the `configure` script automatically sets these parameters, sites can choose to specify additional parameters. If the values selected in `configure` are satisfactory, then this section can be safely ignored.

The parameters needed for proper initial startup include the following:

Parameter	Instructions
SCHEDCFG	The `SCHEDCFG` parameter specifies how the Moab server will execute and communicate with client requests. The `SERVER` attribute enables Moab client commands to locate the Moab server and is specified as a URL or in `<HOST>[:<PORT>]` format. For example: `SCHEDCFG[orion] SERVER=cw.psu.edu`

Parameter	Instructions
	Specifying the server in the Moab configuration file is optional. If nothing is specified, `gethostname()` is called. You can restart Moab and run mdiag -S to confirm that the correct host name is specified. The `SERVER` attribute can also be set using the environment variable `$MOABSERVER`. Using this variable enables you to quickly change to the Moab server that client commands will connect to. `> export MOABSERVER=cluster2:12221`
ADMINCFG	Moab provides role-based security enabled via multiple levels of admin access. Users who are to be granted full control of all Moab functions should be indicated by setting the `ADMINCFG[1]` parameter. The first user in this `USERS` attribute list is considered the *primary* administrator. It is the ID under which Moab will execute. For example, the following can be used to enable users `greg` and `thomas` as level 1 admins: `ADMINCFG[1] USERS=greg,thomas` Moab can only be launched by the primary admin user ID. The primary admin should be configured as a manager/operator/administrator in every resource manager with which Moab will interface. If the msub command will be used, then 'root' *must* be the primary admin. Moab's home directory and contents should be owned by the primary admin.
RMCFG	For Moab to properly interact with a resource manager, the interface to this resource manager must be defined as described in 11.2 Resource Manager Configuration. Further, it is important that the primary Moab Admin also be a resource manager admin within each of those systems. For example, to interface to a Torque resource manager, the following can be used: `RMCFG[torque1] TYPE=pbs`

2.1.2 geometry.cfg

If topology-aware scheduling is part of your configuration, you will need to create the geometry.cfg file and specify compute node locations using NODEGEOMETRY.

Contact your Adaptive Computing account representative to create the geometry.cfg and define NODEGEOMETRY for you system.

Related Topics

- Appendix A: Moab Parameters
- 3.7.4 mdiag (command for diagnosing current Moab configuration)

2.2 Layout of Scheduler Components

In this section:

2.2.1 Layout of Scheduler Components

Moab is initially unpacked into a simple one-deep directory structure. What follows demonstrates the default layout of scheduler components; some of the files (such as log and statistics files) are created while Moab runs.

- `$(MOABHOMEDIR)` Default is `/opt/moab`, which can be modified via the `--with-homedir` parameter during `./configure`. `$(MOABHOMEDIR)` contains the files shown in the table below:

Filename	Description
contrib/	Directory containing contributed code and plug-ins.
.counters	File containing last 3 counters for InsightIDs, jobs, and reservations respectively. Created during installation and required for Moab operation.
docs/	Directory for documentation.
etc/	Directory for configuration files.
lib/	Directory for library files (primarily for `tools/`).

Filename	Description
log/	Directory for log files.
[etc/]moab.cfg	General configuration file (can be located in `$(MOABHOMEDIR)` or `$(MOABHOMEDIR)/etc`).
.moab.ck	Checkpoint file.
[etc/].moab.key	Secret key used in authentication (can be located in `$(MOABHOMEDIR)` or `$(MOABHOMEDIR)/etc`).
moab.dat	Configuration file generated by Moab Cluster Manager.
[etc/]moab-client.cfg	Client configuration file (can be located in `$(MOABHOMEDIR)` or `$(MOABHOMEDIR)/etc`).
moab.lic	License file.
moab.log	Log file.
moab.log.1	Previous log file.
.moab.pid	Lock file.
[etc/]moab-private.cfg	Secure configuration file containing private information (can be located in `$(MOABHOMEDIR)` or `$(MOABHOMEDIR)/etc`).
stats/	Directory for statistics files: ◦ `events.<date>` – event files ◦ `{DAY|WEEK|MONTH|YEAR}.<date>` – usage profiling data ◦ `FS.<PARTITION>.<epochtime>` – fairshare usage data

- `$(MOABINSTDIR)` Default is `/opt/moab`, which can be modified via the `--prefix` parameter during `./configure`. `$(MOABINSTDIR)` contains the files shown in the table below:

Filename	Description
bin/	Directory for client commands (for example, showq, setres, etc.).

Filename	Description
moab	Moab binary.
sbin/	Directory for server daemons.
tools/	Directory for resource manager interfaces and local scripts.

- `/etc/moab.cfg` If the Moab home directory cannot be found at startup, this file is checked to see if it declares the Moab home directory. If a declaration exists, the system checks the declared directory to find Moab. The syntax is: `MOABHOMEDIR=<DIRECTORY>`.

If you want to run Moab from a different directory other than `/opt/moab` but did not use `the --with-homedir` parameter during `./configure`, you can set the `$MOABHOMEDIR` environment variable, declare the home directory in the `/etc/moab.cfg` file, or use the `-C` command line option when using the Moab server or client commands to specify the configuration file location.

When Moab runs, it creates a log file, `moab.log`, in the `log/` directory and creates a statistics file in the `stats/` directory with the naming convention `events.WWW_MMM_DD_YYYY` (for example, `events.Sat_Oct_8_2024`). Additionally, a checkpoint file, `.moab.ck`, and lock file, `.moab.pid`, are maintained in the Moab home directory.

2.2.2 Layout of Scheduler Components with Integrated Database Enabled

If the parameter USEDATABASE INTERNAL is configured, the layout of scheduler components varies slightly. The `.moab.ck` file and usage profiling data (`stat/{DAY|WEEK|MONTH|YEAR}.<date>`) are stored in the `moab.db` database. In addition, the event information is stored in both event files: (`stat/events.<date>`) and `moab.db`.

Related Topics

- Chapter 3: Scheduler Commands

2.3 Scheduling Environment

Moab functions by manipulating a number of elementary objects, including jobs, nodes, reservations, QoS structures, resource managers, and policies. Multiple minor elementary objects and composite objects are also used; these objects are defined in 2.4 Scheduling Dictionary.

In this section:

2.3.1 Jobs

Job information is provided to the Moab scheduler from a resource manager such as PBS or Wiki. Job attributes include ownership of the job, job state, amount and type of resources required by the job, and a wallclock limit indicating how long the resources are required. A job consists of one or more task groups, each of which requests a number of resources of a given type; for example, a job may consist of two task groups, the first asking for a single master task consisting of *1 IBM SP node with at least 512 MB of RAM* and the second asking for a set of slave tasks such as *24 IBM SP nodes with at least 128 MB of RAM*. Each task group consists of one or more tasks where a task is defined as the minimal independent unit of resources. By default, each task is equivalent to 1 processor. In SMP environments, however, users might want to tie one or more processors together with a certain amount of memory and other resources.

In this topic:

2.3.1.A Job States

The job's *state* indicates its current status and eligibility for execution and can be any of the values listed in the following tables:

Table 2-1: Pre-execution states

State	Definition
Deferred	Job that has been held by Moab due to an inability to schedule the job under current conditions. Deferred jobs are held for DEFERTIME before being placed in the idle queue. This process is repeated DEFERCOUNT times before the job is placed in batch hold.
Hold	Job is idle and is not eligible to run due to a user, (system) administrator, or batch system *hold* (also, batchhold, systemhold, userhold).
Idle	Job is currently queued and eligible to run but is not executing (also, **notqueued**).
NotQueued	The job has not been queued.
Unknown	Moab cannot determine the state of the job.

Table 2-2: Execution states

State	Definition
Starting	Batch system has attempted to start the job and the job is currently performing *prestart* tasks that may include provisioning resources, staging data, or executing system pre-launch scripts.
Running	Job is currently executing the user application.
Suspended	Job was running but has been suspended by the scheduler or an admin; user application is still in place on the allocated compute resources, but it is not executing.

Table 2-3: Post-execution states

State	Definition
Completed	Job has completed running without failure.
Removed	Job has run to its requested walltime successfully but has been canceled by the scheduler or resource manager due to exceeding its walltime or violating another policy; includes jobs canceled by users or admins either before or after a job has started.
Vacated	Job canceled after partial execution due to a system failure.

2.3.1.B Task Group (or Req)

A job *task group* (or req) consists of a request for a single type of resources. Each task group consists of the following components:

Component	Description
Task Definition	A specification of the elementary resources that compose an individual task.
Resource Constraints	A specification of conditions that must be met for resource matching to occur. Only resources from nodes that meet *all* resource constraints can be allocated to the job task group.
Task Count	The number of task instances required by the task group.
Task List	The list of nodes on which the task instances are located.
Task Group Statistics	Statistics tracking resource utilization.

2.3.2 Nodes

Moab recognizes a node as a collection of resources with a particular set of associated attributes. This definition is similar to the traditional notion of a node found in a Linux cluster or supercomputer wherein a node is defined as one or more CPUs, associated memory, and possibly other compute resources such as local disk, swap, network adapters, and software licenses. Additionally, this node is described by various attributes such as an architecture type or operating system. Nodes range in size from small uniprocessor PCs to large symmetric multiprocessing (SMP) systems where a single node may consist of hundreds of CPUs and massive amounts of memory.

In many cluster environments, the primary source of information about the configuration and status of a compute node is the resource manager. This information can be augmented by additional information sources including node monitors and information services. Further, extensive node policy and node configuration information can be specified within Moab via the graphical tools or the configuration file. Moab aggregates this information and presents a comprehensive view of the node configuration, usages, and state.

While a node in Moab in most cases represents a standard compute host, nodes can also be used to represent more generalized resources. The GLOBAL node possesses floating resources that are available cluster wide, and created virtual nodes (such as network, software, and data nodes) track and allocate resource usage for other resource types.

For additional node information, see Chapter 10: General Node Administration.

2.3.3 Advance Reservations

An advance reservation dedicates a block of specific resources for a particular use. Each reservation consists of a list of resources, an access control list, and a time range for enforcing the access control list. The reservation ensures the matching nodes are used according to the access controls and policy constraints within the time frame specified. For example, a reservation could reserve 20 processors and 10 GB of memory for users Bob and John from Friday 6:00 A.M. to Saturday 10:00 P.M. Moab uses advance reservations extensively to manage backfill, guarantee resource availability for active jobs, allow service guarantees, support deadlines, and enable metascheduling. Moab also supports both regularly recurring reservations and the creation of dynamic one-time reservations for special needs. Advance reservations are described in detail in 6.1 Advance Reservations.

2.3.4 Policies

A configuration file specifies policies and controls how and when jobs start. Policies include job prioritization, fairness policies, fairshare configuration policies, and scheduling policies.

2.3.5 Resources

Jobs, nodes, and reservations all deal with the abstract concept of a resource. A resource in the Moab world is one of the following:

Resource	Description
processors	Specify with a simple count value
memory	Specify real memory or RAM in megabytes (MB)
swap	Specify virtual memory or *swap* in megabytes (MB)
disk	Specify local disk in megabytes (MB)

In addition to these elementary resource types, there are two higher level resource concepts used within Moab: Task and the processor equivalent [or PE (Processor Equivalent) Calculation], as explained below.

In this topic:

2.3.5.A Task

A task is a collection of elementary resources that must be allocated together within a single node. For example, a task may consist of 1 processor, 512 MB of RAM, and 2 GB of local disk. A key aspect of a task is that the resources associated with the task must be allocated as an atomic unit, without spanning node boundaries. A task requesting 2 processors cannot be satisfied by allocating 2 uniprocessor nodes, nor can a task requesting 1 processor and 1 GB of memory be satisfied by allocating 1 processor on 1 node and memory on another.

In Moab, when jobs or reservations request resources, they do so in terms of tasks typically using a task count and a task definition. By default, a task maps directly to a single processor within a job and maps to a full node within reservations. In all cases, this default definition can be overridden by specifying a new task definition.

Within both jobs and reservations, depending on task definition, it is possible to have multiple tasks from the same job mapped to the same node. For example, a job requesting 4 tasks using the default task definition of 1 processor per task, can be satisfied by 2 dual processor nodes.

2.3.5.B PE (Processor Equivalent) Calculation

The concept of the processor equivalent, or PE, arose out of the need to translate multi-resource consumption requests into a scalar value. It is not an elementary resource but rather a derived resource metric. It is a measure of the actual impact of a set of requested resources by a job on the total resources available system wide. It is calculated as follows:

```
PE = MAX(ProcsRequestedByJob / TotalOnlineProcs,
MemoryRequestedByJob / TotalOnlineMemory,
DiskRequestedByJob / TotalOnlineDisk,
SwapRequestedByJob / TotalOnlineSwap) * TotalOnlineProcs
```

For example, if a job requested 20% of the total processors and 50% of the total memory of a 128-processor MPP system, only two such jobs could be supported by this system. The job is essentially using 50% of all available resources since the system can only be scheduled to its most constrained resource - memory in this case. The processor equivalents for this job should be 50% of the processors, or PE = 64.

Another example: Assume a homogeneous 100-node system with 4 processors and 1 GB of memory per node. A job is submitted requesting 2 processors and 768 MB of memory. The PE for this job is calculated as follows:

```
PE = MAX(2/(100*4), 768/(100*1024)) * (100*4) = 3.
```

This result makes sense since the job will be consuming 3/4 of the memory on a 4-processor node.

The calculation works equally well on homogeneous or heterogeneous systems, uniprocessor or large SMP systems.

2.3.6 Class (or Queue)

A class (or queue) is a logical container object that implicitly or explicitly applies policies to jobs. In most cases, a class is defined and configured within the resource manager and associated with one or more of the following attributes or constraints:

Attribute	Description
Default Job Attributes	A queue can be associated with a default job duration, default size, or default resource requirements.
Host Constraints	A queue can constrain job execution to a particular set of hosts.
Job Constraints	A queue can constrain the attributes of jobs that can be submitted, including setting limits such as max wallclock time and minimum number of processors.
Access List	A queue can constrain who can submit jobs into it based on such things as user lists and group lists.
Special Access	A queue can associate special privileges with jobs including adjusted job priority.

As stated previously, most resource managers allow full class configuration within the resource manager. Where additional class configuration is required, the CLASSCFG parameter can be used.

Moab tracks class usage as a consumable resource allowing sites to limit the number of jobs using a particular class. This is done by monitoring class initiators that may be considered to be a ticket to run in a particular class. Any compute node can simultaneously support several types of classes and any number of initiators of each type. By default, nodes will have a one-to-one mapping between class initiators and configured processors. For every job task run on the node, one class initiator of the appropriate type is consumed. For example, a 3-processor job submitted to the class 'batch' consumes three batch class initiators on the nodes where it runs.

Using queues as consumable resources enables sites to specify various policies by adjusting the class initiator to node mapping. For example, a site running serial jobs might want to allow a particular 8-processor node to run any combination of batch and special jobs subject to the following constraints:

- Only 8 jobs of any type allowed simultaneously.
- No more than 4 special jobs allowed simultaneously.

To enable this policy, the site can set the node's MAXJOB policy to 8 and configure the node with 4 special class initiators and 8 batch class initiators.

In virtually all cases, jobs have a one-to-one correspondence between processors requested and class initiators required. However, this is not a requirement, and with special configuration, sites can choose to associate job tasks with arbitrary combinations of class initiator requirements.

In displaying class initiator status, Moab signifies the type and number of class initiators available using the format [<CLASSNAME>:<CLASSCOUNT>]. This is most commonly seen in the output of node status commands indicating the number of configured and available class initiators, or in job status commands when displaying class initiator requirements.

2.3.7 Resource Manager (RM)

While other systems may have more strict interpretations of a resource manager and its responsibilities, Moab's multi-resource manager support allows a much more liberal interpretation. In essence, any object that can provide environmental information and environmental control can be used as a resource manager, including sources of resource, workload, credential, or policy information such as scripts, peer services, databases, web services, hardware monitors, or even flat files. Likewise, Moab considers to be a resource manager any tool that provides control over the cluster environment whether that be a license manager, queue manager, checkpoint facility, provisioning manager, network manager, or storage manager.

Moab aggregates information from multiple unrelated sources into a larger more complete world view of the cluster that includes all the information and control found within a standard resource manager such as Torque, including node, job, and queue management services. For more information, see Chapter 11: Resource Managers and Interfaces overview.

Arbitrary Resource

Nodes can also be configured to support various arbitrary resources. Use the NODECFG parameter to specify information about such resources. For example, you could configure a node to have *256 MB RAM, 4 processors, 1 GB swap, and 2 tape drives*.

2.4 Scheduling Dictionary

Index: A C D E F G J M N P Q R S T U W

A

Account	
Definition	A credential also known as 'project ID.' Multiple users can be associated a single account ID and each user can have access to multiple accounts. See Credential definition and ACCOUNTCFG parameter.
Example	`ACCOUNT=hgc13`

ACL (Access Control List)	
Definition	In the context of scheduling, an access control list is used and applied much as it is elsewhere. An ACL defines what credentials are required to access or use particular objects. The principal objects to which ACLs are applied are reservations and QoSes. ACLs can contain both allow and deny statements, include wildcards, and contain rules based on multiple object types.
Example	Reservation META1 contains 4 access statements: • Allow jobs owned by user "john" or "bob" • Allow jobs with QoS "premium" • Deny jobs in class "debug" • Allow jobs with a duration of less than 1 hour

Allocation	
Definition	A logical, scalar unit assigned to users on a credential basis, providing access to a particular quantity of compute resources. Allocations are consumed by jobs associated with those credentials.
Example	`ALLOCATION=30000`

C

Class	
Definition	(See Queue) A class is a logical container object that holds jobs allowing you to associate various constraints and defaults to these jobs. Class access can also be tied to individual nodes defining whether a particular node will accept a job associated with a given class. Class based access to a node is denied unless explicitly allowed via resource manager configuration. Within Moab, classes are tied to jobs as a credential.

Class	
Example	job "cw.073" is submitted to class batch node "cl02" accepts jobs in class batch reservation weekend allows access to jobs in class batch

CPU	
Definition	A single processing unit. A CPU is a consumable resource. Nodes typically consist of one or more CPUs (same as processor).

Credential	
Definition	An attribute associated with jobs and other objects that determines object identity. In the case of schedulers and resource managers, credential based policies and limits are often established. At submit time, jobs are associated with a number of credentials such as user, group, account, QoS, and class. These job credentials subject the job to various polices and grant it various types of access. In most cases, credentials set both the privileges of the job and the ID of the actual job executable.

Example	Job "cw.24001" possesses the following credentials:
	`USER=john;GROUP=staff;ACCOUNT=[NONE];`
	`QOS=[DEFAULT];CLASS=batch`

D

Disk	
Definition	A quantity of local disk available for use by batch jobs. Disk is a consumable resource.

E

Execution Environment	
Definition	A description of the environment where the executable is launched. This environment can include attributes such as the following: • an executable • command line arguments • input file

Execution Environment	
	• output file • local user ID • local group ID • process resource limits

Example	Job "cw.24001" possesses the following execution environment:
	`EXEC=/bin/sleep;ARGS="60";`
	`INPUT=[NONE];OUTPUT=[NONE];`
	`USER=loadl;GROUP=staff;`

F

Fairshare	
Definition	A mechanism that allows historical resource utilization information to be incorporated into job priority decisions.

Fairness	
Definition	The access to shared compute resources that each user is granted. Access can be equal or based on factors such as historical resource usage, political issues, and job value.

G

Group	
Definition	A credential typically directly mapping to a user's UNIX group ID.

J

Job	
Definition	The fundamental object of resource consumption. A job contains the following components: • A list of required consumable resources • A list of resource constraints controlling which resources can be allocated to the job • A list of job constraints controlling where, when, and how the job should run

Job	
	• A list of credentials • An execution environment

Job Constraints	
Definition	A set of conditions that must be fulfilled for the job to start. These conditions are far reaching and can include one or more of the following: • When the job can run. After time X, within Y minutes. • Which resources can be allocated. For example, node must possess at least 512 MB of RAM, run only in partition A or Partition C, or run on HostA and HostB. • Starting job relative to a particular event. Start after job X successfully completes.

Example	`RELEASETIME>='Tue Feb 12, 11:00AM'`
	`DEPEND=AFTERANY:cw.2004`
	`NODEMEMORY==256MB`

M

Memory	
Definition	A quantity of physical memory (RAM). Memory is provided by compute nodes. It is required as a constraint or consumed as a consumable resource by jobs. Within Moab, memory is tracked and reported in megabytes (MB).

Example	Node "node001" provides the following resources:
	`PROCS=1,MEMORY=512,SWAP=1024`
	"Job cw.24004" consumes the following resources per task:
	`PROCS=1,MEMORY=256`

N

Node	
Definition	A node is the fundamental object associated with compute resources. Each node contains the following components: • A list of consumable resources • A list of node attributes

Node Attribute	
Definition	A node attribute is a non-quantitative aspect of a node. Attributes typically describe the node itself or possibly aspects of various node resources such as processors or memory. While it is probably not optimal to aggregate node and resource attributes together in this manner, it is common practice. Common node attributes include processor architecture, operating system, and processor speed. Jobs often specify that resources be allocated from nodes possessing certain node attributes.
Example	`ARCH=AMD,OS=LINUX24,PROCSPEED=950`

Node Feature	
Definition	A node feature is a node attribute that is typically specified locally via a configuration file. Node features are opaque strings associated with the node by the resource manager that generally only have meaning to the end-user, or possibly to the scheduler. A node feature is commonly associated with a subset of nodes allowing end-users to request use of this subset by requiring that resources be allocated from nodes with this feature present. In many cases, node features are used to extend the information provided by the resource manager.
Example	`FEATURE=s950,pIII,geology` *This can be used to indicate that the node possesses a 950 MHz Pentium III processor and that the node is owned by the Geology department.*

P

Processor	
Definition	A processing unit. A processor is a consumable resource. Nodes typically consist of one or more processors (same as CPU).

Q

Quality of Service (QoS)	
Definition	An object that provides special services, resources, and so forth.

Queue	
Definition	(see Class)

R

Reservation	
Definition	An object that reserves a specific collection or resources for a specific timeframe for use by jobs that meet specific conditions. Each reservation consists of three major components: (1) a set of resources, (2) a time frame, and (3) an access control list. It is a scheduler role to ensure that the access control list is not violated during the reservation's lifetime (that is, its time frame) on the resources listed. For example, a reservation may specify that node002 is reserved for user Tom on Friday. The scheduler is therefore constrained to make certain that only Tom's jobs can use node002 at any time on Friday.
Example	Reserve 24 processors and 8 GB of memory from time T1 to time T2 for use by user X or jobs in the class batch.

Resource	
Definition	Hardware, generic resources such as software, and features available on a node, including memory, disk, swap, and processors.

Resource, Available	
Definition	A compute node's configured resources minus the *maximum* of the sum of the resources utilized by all job tasks running on the node and the resources dedicated; that is, R.Available = R.Configure - MAX(R.Dedicated,R.Utilized). In most cases, resources utilized will be associated with compute jobs that the batch system has started on the compute nodes, although resource consumption may also come from the operating system or *rogue* processes outside of the batch system's knowledge or control. Further, in a well-managed system, utilized resources are less than or equal to dedicated resources and when exceptions are detected, one or more usage-based limits are activated to preempt the jobs violating their requested resource usage.
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. 1 processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Resources available to user jsmith on node "cl003": • 2 processors

Resource, Available	
	• 392 MB memory Resources available to a user other than jsmith on node "cl003": • 1 processor • 142 MB memory

Resource, Configured	
Definition	The total amount of consumable resources that are available on a compute node for use by job tasks.
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. 1 processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Configured resources for node "cl003": • 4 processors • 512 MB memory

Resource, Consumable	
Definition	Any object that can be used (that is, consumed and therefore made unavailable to another job) by, or dedicated to a job is considered to be a resource. Common examples of resources are a node's physical memory or local disk. As these resources can be given to one job and therefore become unavailable to another, they are considered to be consumable. Other aspects of a node, such as its operating system, are not considered to be consumable since its use by one job does not preclude its use by another. Note that some node objects, such as a network adapter, can be dedicated under some operating systems and resource managers and not under others. On systems where the network adapter cannot be dedicated and the network usage per job cannot be specified or tracked, network adapters are not considered to be resources, but rather attributes. Nodes possess a specific quantity of consumable resources such as real memory, local disk, or processors. In a resource management system, the node manager can choose to report only those configured resources available to batch jobs. For example, a node may possess an 80 GB hard drive but may have only 20 GB dedicated to batch jobs. Consequently, the resource manager may report that the node has 20 GB of local disk available when idle. Jobs can explicitly request a certain quantity of consumable resources.

Resource, Constraint	
Definition	A resource constraint imposes a rule on which resources can be used to match a

Resource, Constraint	
	resource request. Resource constraints either specify a required quantity and type of resource or a required node attribute. All resource constraints must be met by any given node to establish a match.

Resource, Dedicated	
Definition	A job may request that a block of resources be dedicated while the job is executing. At other times, a certain number of resources can be reserved for use by a particular user or group. In these cases, the scheduler is responsible for guaranteeing that these resources, utilized or not, are set aside and made unavailable to other jobs.
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. 1 processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Dedicated resources for node "cl003": • 1 processor • 250 MB memory

Resource, Utilized	
Definition	All consumable resources actually used by all job tasks running on the compute node.
Example	Node "cl003" has 4 processors and 512 MB of memory. It is executing 2 tasks of job "clserver.0041" that are using 1 processor and 60 MB of memory each. 1 processor and 250 MB of memory are reserved for user "jsmith" but are not currently in use. Utilized resources for node "cl003": • 2 processors • 120 MB memory

S

Swap	
Definition	A quantity of virtual memory available for use by batch jobs. Swap is a consumable resource provided by nodes and consumed by jobs.

T

Task	
Definition	An atomic collection of consumable resources.

Time to Live (TTL)	
Definition	Specifies the time that a node is supposed to be retired by Moab. Moab will not schedule any jobs on a node after its time to live has passed.

U

User, Global	
Definition	The user credential used to provide access to functions and resources. In local scheduling, global user IDs map directly to local user IDs.

User, Local	
Definition	The user credential under which the job executable will be launched.

W

Workload	
Definition	A set of tasks to be performed or services to be provided by a system that comprises a set of resources.

2.5 Scheduling Iterations and Job Flow

In this section:

2.5.1 Scheduling Iterations

In any given scheduling iteration, many activities take place, examples of which are listed below:

- Update State Information
- Handle User Requests
- Perform Next Scheduling Cycle

2.5.1.A Update State Information

Each iteration, the scheduler contacts the resource manager(s) and requests up-to-date information on compute resources, workload, and policy configuration. On most systems, these calls are to a centralized resource manager daemon that possesses all information. Jobs can be reported as being in any of the following states listed in the job state table.

2.5.1.B Handle User Requests

User requests include any call requesting state information, configuration changes, or job or resource manipulation commands. These requests can come in the form of user client calls, peer daemon calls, or process signals.

2.5.1.C Perform Next Scheduling Cycle

Moab operates on a polling/event driven basis. When all scheduling activities complete, Moab processes user requests until a new resource manager event is received or an internal event is generated. Resource manager events include activities such as a new job submission or completion of an active job, addition of new node resources, or changes in resource manager policies. Internal events include administrator schedule requests, reservation activation/deactivation, or the expiration of the RMPOLLINTERVAL timer.

2.5.2 Detailed Job Flow

In this topic:

2.5.2.A Determine Basic Job Feasibility

The first step in scheduling is determining which jobs are feasible. This step eliminates jobs that have job holds in place, invalid job states (such as Completed, Not Queued, Deferred), or unsatisfied preconditions. Preconditions can include stage-in files or completion of preliminary job steps.

2.5.2.B Prioritize Jobs

With a list of feasible jobs created, the next step involves determining the relative priority of all jobs within that list. A priority for each job is calculated based on job attributes such as job owner, job size, and length of time the job has been queued.

2.5.2.C Enforce Configured Throttling Policies

Any configured throttling policies are then applied constraining how many jobs, nodes, processors, and so forth are allowed on a per credential basis. Jobs that violate these policies are not considered for scheduling.

2.5.2.D Determine Resource Availability

For each job, Moab attempts to locate the required compute resources needed by the job. For a match to be made, the node must possess all node attributes specified by the job and possess adequate available resources to meet the 'TasksPerNode' job constraint. (Default 'TasksPerNode' is 1.) Normally, Moab determines that a node has adequate resources if the resources are *neither utilized by nor dedicated to* another job using the calculation.

R.Available = R.Configured - MAX(R.Dedicated,R.Utilized).

The NODEAVAILABILITYPOLICY parameter can be modified to adjust this behavior.

2.5.2.E Allocate Resources to Job

If adequate resources can be found for a job, the node allocation policy is then applied to select the best set of resources. These allocation policies allow selection criteria such as speed of node, type of reservations, or excess node resources to be figured into the allocation decision to improve the performance of the job and maximize the freedom of the scheduler in making future scheduling decisions.

2.5.2.F Launch Job

With the resources selected and task distribution mapped, the scheduler then contacts the resource manager and informs it where and how to launch the job. The resource manager then initiates the actual job executable.

2.6 Configuring the Scheduler

Scheduler configuration is maintained using the flat text configuration file `moab.cfg`. All configuration file entries consist of simple `<PARAMETER>` `<VALUE>` pairs that are whitespace delimited. Parameter names are not case sensitive but `<VALUE>` settings are. Some parameters are array values and should be specified as `<PARAMETER>[<INDEX>]` (Example: `QOSCFG[hiprio] PRIORITY=1000`); the `<VALUE>` settings can be integers, floats, strings, or arrays of these. Some parameters can be specified as arrays wherein index values can be numeric or alphanumeric strings. If no array index is specified for an array parameter, an index of zero (0) is assumed. The example below includes both array based and non-array based parameters:

```
SCHEDCFG[cluster2] SERVER=head.c2.org MODE=NORMAL
LOGLEVEL 6
LOGDIR   /var/tmp/moablog
```

See Appendix A: Moab Parameters for information on specific parameters.

The `moab.cfg` file is read when Moab is started up or recycled. Also, the mschedctl -m command can be used to reconfigure the scheduler at any time, updating some or all of the configurable parameters dynamically. This command can be used to modify parameters either permanently or temporarily. For example, the command *`mschedctl -m LOGLEVEL 3`*will temporarily adjust the scheduler log level. When the scheduler restarts, the log level restores to the value stored in the Moab configuration files. To adjust a parameter permanently, the option `--flags=persistent` should be set.

At any time, the current server parameter settings can be viewed using the mschedctl -l command.

Adjusting Server Behavior

Most aspects of Moab behavior are configurable. This includes both scheduling policy behavior and daemon behavior. In terms of configuring server behavior, the following realms are most commonly modified.

Logging

Moab provides extensive and highly configurable logging facilities controlled by parameters.

Parameter	Description
LOGDIR	Indicates directory for log files.
LOGFACILITY	Indicates scheduling facilities to track.
LOGFILE	Indicates path name of log file.
LOGFILEMAXSIZE	Indicates maximum size of log file before rolling.
LOGFILEROLLDEPTH	Indicates maximum number of log files to maintain.
LOGLEVEL	Indicates verbosity of logging.

Checkpointing

Moab checkpoints its internal state. The checkpoint file records statistics and attributes for jobs, nodes, reservations, users, groups, classes, and almost every other scheduling object.

Parameter	Description
CHECKPOINTEXPIRATIONTIME	Indicates how long unmodified data should be kept after the associated object has disappeared; that is, job priority for a job no longer detected.
CHECKPOINTFILE	Indicates path name of checkpoint file.
CHECKPOINTINTERVAL	Indicates interval between subsequent checkpoints.

Client Interface

Clients will read from the client configuration file (`moab-client.cfg`), if present, and then from the server configuration file (`moab.cfg`), if present. First, clients will search for the presence of a `moab-client.cfg` file, loading client parameters from the first file detected in `$MOABHOMEDIR` or `$MOABHOMEDIR/etc`. Next, clients will search for the presence of a `moab.cfg` file, loading client parameters from the first file detected in `$MOABHOMEDIR` or `$MOABHOMEDIR/etc`, overriding any parameter values read from the client configuration file. If both files are present on a host, it is safe to remove the `moab-client.cfg` file after merging the client parameters into the `moab.cfg` file.

The Client interface is configured using the SCHEDCFG parameter. Most commonly, the attributes `SERVER` and `PORT` must be set to point client commands to the appropriate Moab server. Other parameters such as CLIENTTIMEOUT can also be set.

By default, Moab listens on all the interfaces of the machine on which it is installed. To bind Moab to a specific address use '`SCHEDCFG[] BINDADDRESS=<IPV4>`' and specify the specific IPv4 address of the interface on which Moab should listen. By default, Moab also verifies that the `SERVER` parameter matches the output of the '`gethostbyname`' system call. To configure Moab to use a different alias (on multi-homed hosts for example) you can specify the valid server aliases using '`SCHEDCFG[] SERVERALIAS=<alias1>,<alias2>...`'

Scheduler Mode

The scheduler mode of operation is controlled by setting the `MODE` attribute of the SCHEDCFG parameter. The following modes are allowed:

Mode	Description
INTERACTIVE	Moab interactively confirms each scheduling action before taking any steps. See 14.1.1.C INTERACTIVE Mode for more information.
MONITOR	Moab observes cluster and workload performance, collects statistics, interacts with allocation management services, and evaluates failures, but it does not actively alter the cluster, including workload scheduling, and resource provisioning. See 14.1.1.A MONITOR Mode for more information.
NORMAL	Moab actively schedules workload according to mission objectives and policies; it creates reservations; starts, cancels, preempts, and modifies jobs; and takes other scheduling actions.
SINGLESTEP	Moab behaves as in NORMAL mode but will only schedule a single iteration and then exit.
SLAVE	Moab behaves as in NORMAL mode but will only start a job when explicitly requested by a trusted grid peer service or admin.
TEST	Moab behaves as in NORMAL mode, will make reservations, and scheduling decisions, but will then only log scheduling actions it would have taken if running in NORMAL mode. In most cases, 'TEST' mode is identical to MONITOR mode. See 14.1.1.B TEST Mode for more information.

Configuring a Job ID Offset

Moab assigns job IDs as integers in numeric order as jobs are submitted, starting with 1. In some situations, you might want to offset the integer at which Moab starts to assign job IDs in your system.

This example describes how to offset the job IDs in a compound system consisting of Site A, Site B, and Site C, each of which runs its own instance of Moab. Users belonging to any of the sites can submit jobs to their own site and to the other two. To simplify aggregation of usage records from the three sites, offset the job IDs for Site B to a starting value higher than the expected total lifetime value for the system; in this example, to `20000000`.

Likewise, set Site C to 20,000,000 more, or 40000000. To do so, set the MINJOBID attribute of SCHEDCFG in each system's moab.cfg to the offset value. To ensure that Moab will never use the same job ID for two different sites, also set MAXJOBID. If the Moab job naming process ever reaches the MAXJOBID, it will start over again with the MINJOBID.

```
SCHEDCFG[moab] SERVER=moab_siteA:4244 MAXJOBID=19999999
```

```
SCHEDCFG[moab] SERVER=moab_siteB:4344 MINJOBID=20000000 MAXJOBID=39999999
```

```
SCHEDCFG[moab] SERVER=moab_siteC:4444 MINJOBID=40000000 MAXJOBID=59999999
```

When users submit jobs to Moab using msub, Moab selects the job ID in numeric order, starting with 1 in Site A, 20000000 in Site B, and 40000000 in Site C.

If the compound system in this example uses Torque as its resource manager and users submit jobs directly to Torque using qsub, Torque assigns the job ID instead of Moab. In this case, you should also offset the Torque job IDs by setting the next_job_number server parameter of Site B and Site C to 20000000 and 40000000, respectively.

```
$user qmgr "set server next_job_number=20000000"
```

```
$user qmgr "set server next_job_number=40000000"
```

Torque job ID limits will allow you to use the 20,000,000 offset scheme for up to 4 sites.

Related Topics

- 2.1 Initial Moab Configuration
- Adding #INCLUDE files to moab.cfg

2.7 Credentials

Moab supports the concept of credentials, which provide a means of attributing policy and resource access to entities such as users and groups. These credentials allow specification of job ownership, tracking of resource usage, enforcement of policies, and many other features. There are five types of credentials: user, group, account, class, and QoS. While the credentials have many similarities, each plays a slightly different role.

In this section:

2.7.1 General Credential Attributes

Internally, credentials are maintained as objects. Credentials can be created, destroyed, queried, and modified. They are associated with jobs and requests providing access and privileges. Each credential type has the following attributes:

- Priority Settings
- Usage Limits
- Service Targets
- Credential and Partition Access
- Statistics
- Credential Defaults, State and Configuration Information

All credentials represent a form of identity, and when applied to a job, express ownership. Consequently, jobs are subject to policies and limits associated with their owners.

In this topic:

2.7.1.A Credential Priority Settings/Priority Weight Offsets

Each credential can be assigned a priority using the `PRIORITY` attribute. This priority affects a job's total credential priority factor as described in the section 4.1.2 Job Priority Factors. In addition, each credential can also specify priority weight offsets, which adjust priority weights that apply to associated jobs. These priority weight offsets include FSWEIGHT (see 5.3.2.C Priority-Based Fairshare for more information), QTWEIGHT, and XFWEIGHT.

For example:

```
# set priority weights
CREDWEIGHT      1
USERWEIGHT      1
CLASSWEIGHT     1
SERVICEWEIGHT   1
XFACTORWEIGHT   10
QUEUETIMEWEIGHT 1000
# set credential priorities
USERCFG[john] PRIORITY=200
CLASSCFG[batch] PRIORITY=15
CLASSCFG[debug] PRIORITY=100
QOSCFG[bottomfeeder] QTWEIGHT=-50 XFWEIGHT=100
ACCOUNTCFG[topfeeder] PRIORITY=100
```

2.7.1.B Credential Usage Limits

Usage limits constrain which jobs can run, which jobs can be considered for scheduling, and what quantity of resources each individual job can consume. With usage limits, policies such as MAXJOB, MAXNODE, and MAXMEM can be enforced against both idle and active jobs. Limits can be applied in any combination as shown in the example below where usage limits include 32 active processors per group and 12 active jobs for user `john`. For a job to run, it must satisfy the most limiting policies of all associated credentials. The Throttling Policy section documents credential usage limits in detail.

```
GROUPCFG[DEFAULT] MAXPROC=32 MAXNODE=100
GROUPCFG[staff]   MAXNODE=200
USERCFG[john]     MAXJOB=12
```

2.7.1.C Service Targets

Credential service targets allow jobs to obtain special treatment to meet usage or response time based metrics. Additional information about service targets can be found in 5.3 Fairshare.

2.7.1.D Credential and Partition Access

Access to partitions and to other credentials can be specified on a per credential basis with credential access lists, default credentials, and credential membership lists.

Credential Access Lists

You can use the `ALIST`, `PLIST`, and `QLIST` attributes (shown in the following table) to specify the list of credentials or partitions that a given credential can access:

Credential	Attribute
Account	`ALIST` (allows credential to access specified list of accounts)
Partition	`PLIST` (allows credential to access specified list of partitions)
QoS	QLIST (allows credential to access specified list of QoSes)

For example:

```
USERCFG[bob]   ALIST=jupiter,quantum
USERCFG[steve] ALIST=quantum
```

Account-based access lists are only enforced if using an accounting manager or if the ENFORCEACCOUNTACCESS parameter is set to TRUE.

Assigning Default Credentials

Use the `*DEF` attribute (shown in the following table) to specify the default credential or partition for a particular credential:

Credential	Attribute
Account	`ADEF` (specifies default account)
Class	`CDEF` (specifies default class)
QoS	`QDEF` (specifies default QoS)

For example:

```
# user bob can access accounts a2, a3, and a6. If no account is explicitly requested,
# his job will be assigned to account a3
USERCFG[bob]   ALIST=a2,a3,a6 ADEF=a3
# user steve can access accounts a14, a7, a2, a6, and a1. If no account is explicitly
# requested, his job will be assigned to account a2
USERCFG[steve] ALIST=a14,a7,a2,a6,a1 ADEF=a2
```

Specifying Credential Membership Lists

As an alternative to specifying access lists, admins can also specify membership lists. This allows a credential to specify who can access it rather than allowing each credential to specify which credentials it can access. Membership lists are controlled using the `MEMBERULIST`, `EXCLUDEUSERLIST` and `REQUIREDUSERLIST` attributes, shown in the following table:

Credential	Attribute
User	---
Account, Group, QoS	`MEMBERULIST`
Class	EXCLUDEUSERLIST and REQUIREDUSERLIST

For example:

```
# account omega3 can only be accessed by users johnh, stevek, jenp
ACCOUNTCFG[omega3] MEMBERULIST=johnh,stevek,jenp
```

Example 2-1: Controlling Partition Access on a Per User Basis

A site may specify the user `john` can access partitions `atlas`, `pluto`, and `zeus` and will default to partition `pluto`. To do this, include the following line in the configuration file:

```
USERCFG[john] PLIST=atlas,pluto,zeus
```

Example 2-2: Controlling QoS Access on a Per Group Basis

A site may also choose to allow everyone in the group `staff` to access QoS `standard` and `special` with a default QoS of `standard`. To do this, include the following line in the configuration file:

```
GROUPCFG[staff] QLIST=standard,special QDEF=standard
```

Example 2-3: Controlling Resource Access on a Per Account Basis

An organization wants to allow everyone in the account `omega3` to access nodes 20 through 24. To do this, include the following in the configuration file:

```
ACCOUNTCFG[omega3] MEMBERULIST=johnh,stevek,jenp
SRCFG[omega3]   HOSTLIST=r:20-24 ACCOUNTLIST=omega3
```

2.7.1.E Credential Statistics

Full statistics are maintained for each credential instance. These statistics record current and historical resource usage, level of service delivered, accuracy of requests, and many other aspects of workload. Note, though, that you must explicitly enable credential statistics as they are not tracked by default. You can enable credential statistics by including the following in the configuration file:

```
USERCFG[DEFAULT]        ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]       ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]     ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]       ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]         ENABLEPROFILING=TRUE
```

2.7.1.F Job Defaults, Credential State, and General Configuration

Credentials can apply defaults and force job configuration settings via the following parameters:

COMMENT	
Description	Associates a comment string with the target credential.

Example	`USERCFG[steve] COMMENT='works for boss, provides good service'`
	`CLASSCFG[i3]   COMMENT='queue for I/O intensive workload'`

HOLD	
Description	Specifies a hold should be placed on all jobs associated with the target credential. Any job associated with the target credential will remain in the hold state (i.e., the only way to remove the hold is to remove or disassociate the target credential from the job). The order in which this HOLD attribute is evaluated depends on the following credential precedence: USERCFG, GROUPCFG, ACCOUNTCFG, CLASSCFG, QOSCFG, USERCFG[DEFAULT], GROUPCFG[DEFAULT], ACCOUNTCFG[DEFAULT], CLASSCFG[DEFAULT], QOSCFG [DEFAULT].

Example	`USERCFG[user1] HOLD=false`
	`GROUPCFG[user1] HOLD=true`
	Moab evaluates the user hold first, sees that it should not put a hold on the job, and moves on with scheduling.
	`GROUPCFG[user1] HOLD=true`
	`CLASSCFG[user1] HOLD=false`
	Moab evaluates the group first, puts a hold on the job, and moves on.

JOBFLAGS	
Description	Assigns the specified job flag to all jobs with the associated credential.

Example	`CLASSCFG[batch] JOBFLAGS=suspendable`
	`QOSCFG[special] JOBFLAGS=restartable`

NOSUBMIT	
Description	Specifies whether jobs belonging to this credential can submit jobs using *msub*.

Example	`ACCOUNTCFG[general]  NOSUBMIT=TRUE`
	`CLASSCFG[special]    NOSUBMIT=TRUE`

OVERRUN	
Description	The amount of time a job can exceed its wallclock limit before being terminated. (Only applies to user and class credentials.)
Example	`CLASSCFG[bigmem] OVERRUN=00:15:00`

VARIABLE	
Description	Specifies attribute-value pairs associated with the specified credential. These variables can be used in triggers and other interfaces to modify system behavior.
Example	`GROUPCFG[staff] VARIABLE='nocharge=true'`

Credentials can carry additional configuration information. They can specify that detailed statistical profiling should occur, that submitted jobs should be held, or that corresponding jobs should be marked as preemptible.

2.7.2 User Credential

The user credential is the fundamental credential within a workload manager; each job requires an association with exactly one user. In fact, the user credential is the only required credential in Moab; all others are optional. In most cases, the job's user credential is configured within or managed by the operating system itself, although Moab can be configured to obtain this information from an independent security and identity management service.

As the fundamental credential, the user credential has a number of unique attributes:

- Role
- Privileges
- Email Address
- Disable Moab User Email
- Disable Memory Enforcement in RESOURCELIMITPOLICY

2.7.2.A Role

Moab supports role-based authorization, mapping particular roles to collections of specific users. See Appendix E: Security for more information.

2.7.2.B Privileges

Moab supports the ability to configure which 'mdiag' commands a user can run.

Give all users as default:

```
USERCFG[DEFAULT] PRIVILEGES=RM:diagnose;NODE:diagnose
```

Users without any specific PRIVILEGES can run 'mdiag -R' and 'mdiag -n'.

Give specific PRIVILEGES:

```
USERCFG[carol]  PRIVILEGES=SCHED:diagnose;NODE:diagnose
```

User 'carol' can run 'mdiag -S' and 'mdiag -n' but NOT 'mdiag -R'.

2.7.2.C Email Address

Facilities exist to allow user notification in the event of job or system failures or under other general conditions. This attribute allows these notifications to be mailed directly to the target user.

```
USERCFG[sally] EMAILADDRESS=sally@acme.com
```

2.7.2.D Disable Moab User Email

You can disable Moab email notifications for a specific user:

```
USERCFG[john] NOEMAIL=TRUE
```

2.7.2.E Disable Memory Enforcement in RESOURCELIMITPOLICY

You can disable memory enforcement for a specific user:

```
USERCFG[doug] FLAGS=DisableMemEnforcement
```

2.7.3 Group Credential

The group credential represents an aggregation of users. User-to-group mappings are often specified by the operating system or resource manager and typically map to a user's UNIX group ID. However, user-to-group mappings can also be provided by a security and identity management service, or you can specify such directly within Moab.

With many resource managers such as Torque and PBSPro, the group associated with a job is either the user's active primary group as specified within the operating system or a group that is explicitly requested at job submission time. When a secondary group is requested, the user's default group and associated policies are not taken into account. Also note that a job can only run under one group. If more constraining policies are required for these systems, an alternative aggregation scheme such as the use of Account or QOS credentials is recommended.

To enable support for secondary groups, add a SCHEDCFG line to moab.cfg with FLAGS=EXTENDEDGROUPSUPPORT.

To submit a job as a secondary group, refer to your local resource manager's job submission options. For Torque users, see the `group_list=g_list` option of the qsub -W command.

2.7.4 Account (or Project) Credential

The account credential is also referred to as the project credential. This credential is generally associated with a group of users along the lines of a particular project for accounting and billing purposes. User-to-accounting mapping can be obtained from a resource manager or accounting manager, or you can configure it directly within Moab. Access to an account can be controlled via the `ALIST` and `ADEF` credential attributes specified via the Identity Manager or the `moab.cfg` file.

The MANAGERS attribute (applicable only to the account and class credentials) allows an admin to assign a user the ability to manage jobs inside the credential, as if the user is the job owner.

Example 2-4: MANAGERS Attribute

```
ACCOUNTCFG[general]  MANAGERS=ops
ACCOUNTCFG[special]  MANAGERS=stevep
```

If a user is able to access more than one account, the desired account can be specified at job submission time using the resource-manager specific attribute. For example, with Torque this is accomplished using the `-A` argument to the `qsub` command.

Example 2-5: Enforcing Account Usage

Job-to-account mapping can be enforced using the `ALIST` attribute and the ENFORCEACCOUNTACCESS parameter:

```
USERCFG[john]    ALIST=proj1,proj3
USERCFG[steve]   ALIST=proj2,proj3,proj4
USERCFG[brad]    ALIST=proj1
USERCFG[DEFAULT] ALIST=proj2
ENFORCEACCOUNTACCESS TRUE
...
```

2.7.5 Class (or Queue) Credential

In this topic:

The concept of the class credential is derived from the resource manager class or queue object. Classes differ from other credentials in that they more directly impact job attributes. In standard HPC usage, a user submits a job to a class and this class imposes a number of factors on the job. The attributes of a class can be specified within the resource manager or directly within Moab.

Class attributes include the following:

- Job Defaults
- Per Job Min/Max Limits
- Resource Access Constraints
- Class Membership Constraints
- Attributes Enabling Class Access to Other Credentials
- Special Class Attributes

For all classes configured in Moab, a resource manager queue with the same name should be created.

When Torque reports a new queue to Moab, a class of the same name is automatically applied to all nodes (the same goes for existing queues when adding nodes). To associate nodes to only specific classes, add `CLASSCFG` entries for every Torque queue, and define the nodes linked to each queue/class via `HOSTLIST` expressions and/or `REMAPCLASS`. (This augments the optional `resources_default.neednodes` queue setting in `qmgr`.)

2.7.5.A Class Job Defaults

Classes can be assigned to a default job template that can apply values to job attributes not explicitly specified by the submitter. Additionally, you can specify shortcut attributes from the table below:

Attribute	Description
DEFAULT.ATTR	Job Attribute
DEFAULT.DISK	Required Disk (in MB)
DEFAULT.EXT	Job Resource Manager Extension
DEFAULT.FEATURES	Required Node Features/Properties
DEFAULT.GRES	Required Consumable Generic Resources
DEFAULT.MEM	Required Memory/RAM (in MB)
DEFAULT.NODESET	Node Set Specification

Attribute	Description
DEFAULT.PROC	Required Processor Count
DEFAULT.TPN	Tasks Per Node
DEFAULT.WCLIMIT	Wallclock Limit

Defaults set in a class/queue of the resource manager will override the default values of the corresponding class/queue specified in Moab.

The parameter RESOURCELIMITPOLICY must be configured in order for the CLASSCFG limits to take effect.

For example:

```
CLASSCFG[batch] DEFAULT.DISK=200MB DEFAULT.FEATURES=prod DEFAULT.WCLIMIT=1:00:00
CLASSCFG[debug] DEFAULT.FEATURES=debug DEFAULT.WCLIMIT=00:05:00
```

2.7.5.B Per Job Min/Max Limits

Classes can be assigned a minimum and a maximum job template that constrains resource requests. Jobs submitted to a particular queue must meet the resource request constraints of these templates. If a job submission exceeds these limits, the entire job submission fails.

Limit	Description
MAX.ARRAYSUBJOBS	Max Allowed Jobs in an Array
MAX.CPUTIME	Max Allowed Utilized CPU Time
MAX.NODE	Max Allowed Node Count
MAX.PROC	Max Allowed Processor Count
MAX.PS	Max Requested Processor-Seconds
MIN.NODE	Min Allowed Node Count
MIN.PROC	Min Allowed Processor Count

Limit	Description
MIN.PS	Min Requested Processor-Seconds
MIN.TPN	Min Tasks Per Node
MIN.WCLIMIT	Min Requested Wallclock Limit
MAX.WCLIMIT	Max Requested Wallclock Limit

The parameters listed in the preceding table are for classes and PARCFG only, not users, accounts, groups or QoSes, and they function on a per-job basis. The `MAX.*` and `MIN.*` parameters are different from the `MAXJOB`, `MAXNODE`, and `MAXMEM` parameters described earlier in the section Credential Usage Limits.

2.7.5.C Resource Access

Classes can be associated with a particular set of compute resources. Consequently, jobs submitted to a given class can only use listed resources. This can be handled at the resource manager level or via the `CLASSCFG HOSTLIST` attribute.

2.7.5.D Class Membership Constraints

Classes can be configured at either the resource manager or scheduler level to only allow select users and groups to access them. Jobs that do not meet these criteria are rejected. If specifying class membership/access at the resource manager level, see the respective resource manager documentation. Moab automatically detects and enforces these constraints. If specifying class membership/access at the scheduler level, use the `REQUIREDUSERLIST` or `EXCLUDEUSERLIST` attributes of the `CLASSCFG` parameter.

Under most resource managers, jobs must always be a member of one and only one class.

2.7.5.E Attributes Enabling Class Access to Other Credentials

Classes can be configured to allow jobs to access other credentials such as QoSes and Accounts. This is accomplished using the `QDEF`, `QLIST`, `ADEF`, and `ALIST` attributes.

2.7.5.F Special Class Attributes

The class object also possesses a few unique attributes including `JOBPROLOG`, `JOBEPILOG`, `RESFAILPOLICY`, and `DISABLEAM` attributes described below.

MANAGERS

Users listed via the `MANAGERS` parameter are granted full control over all jobs submitted to or running within the specified class:

```
# allow john and steve to cancel and modify all jobs submitted to the class/queue
special
CLASSCFG[special] MANAGERS=john,steve
```

In particular, a class manager can perform the following actions on jobs within a class/queue:

- view/diagnose job (checkjob)
- cancel, requeue, suspend, resume, and checkpoint job (mjobctl)
- modify job (mjobctl)

JOBPROLOG

The `JOBPROLOG` class performs a function similar to the resource manager level job prolog feature; however, there are some key differences:

- Moab prologs execute on the head node; resource manager prologs execute on the nodes allocated to the job.
- Moab prologs execute as the primary Moab Admin, resource manager prologs execute as root.
- Moab prologs can incorporate cluster environment information into their decisions and actions. See valid epilog and prolog variables below.
- Unique Moab prologs can be specified on a per class basis.
- Job start requests are not sent to the resource manager until the Moab job prolog is successfully completed.
- Error messages generated by a Moab prolog are attached to jobs and associated objects; stderr from prolog script is attached to job.
- Moab prologs have access to Moab internal and peer services.

Valid epilog and prolog variables are:

Variable	Description
$TIME	Time that the trigger launches
$HOME	Moab home directory
$USER	User name the job is running under
$JOBID	Unique job identifier
$HOSTLIST	Entire host list for job
$MASTERHOST	Master host for job

The `JOBPROLOG` class attribute enables you to specify a unique per-class action to take before a job is allowed to start. This can be used for environmental provisioning, pre-execution resource checking, security management, and other functions. Sample uses may include enabling a VLAN, mounting a global file system, installing a new application or virtual node image, creating dynamic storage partitions, or activating job specific software services.

A prolog is considered to have failed if it returns a negative number. If a prolog fails, the associated job will not start.

If a prolog executes successfully, the associated epilog is guaranteed to start, even if the job fails for any reason. This allows the epilog to undo any changes made to the system by the prolog.

Job Prolog Examples

```
# explicitly specify prolog arguments for special epilog
CLASSCFG[special] JOBPROLOG='$TOOLSDIR/specialprolog.pl $JOBID $HOSTLIST'
# use default prolog arguments for batch prolog
CLASSCFG[batch]   JOBPROLOG=$TOOLSDIR/batchprolog.pl
```

JOBEPILOG

The Moab epilog is nearly identical to the prolog in functionality except that it runs after the job completes within the resource manager but before the scheduler releases the allocated resources for use by subsequent jobs. It is commonly used for job clean-up, file transfers, signaling peer services, and undoing other forms of resource customization.

An epilog is considered to have failed if it returns a negative number. If an epilog fails, the associated job will be annotated and a message will be sent to admins.

RESFAILPOLICY

This policy allows specification of the action to take on a per-class basis when a failure occurs on a node allocated to an actively running job. See 4.4 Node Availability Policies for more information.

DISABLEAM

You can disable allocation management for jobs in specific classes by setting the `DISABLEAM` class attribute to `TRUE`. For all jobs outside of the specified classes, allocation enforcement will continue to be enforced.

```
# do not enforce allocations on low priority and debug jobs
CLASSCFG[lowprio]  DISABLEAM=TRUE
CLASSCFG[debug]    DISABLEAM=TRUE
```

2.7.5.G Setting Default Classes

In many cases, end-users do not want to be concerned with specifying a job class/queue. This is often handled by defining a default class. Whenever a user does not explicitly submit a job to a particular class, a default class, if specified, is used. In resource managers such as Torque, this can be done at the resource manager level and its impact is transparent to the scheduler. The default class can also be enabled within the scheduler on a per resource manager or per user basis. To set a resource manager default class within Moab, use the `DEFAULTCLASS` attribute of the RMCFG parameter. For per user defaults, use the `CDEF` attribute of the USERCFG parameter.

2.7.5.H Creating a Remap Class

If a single default class is not adequate, Moab provides more flexible options with the REMAPCLASS parameter. If this parameter is set and a job is submitted to the remap class, Moab attempts to determine the final class to which a job belongs based on the resources requested. If a remap class is specified, Moab compares the job's requested nodes, processors, memory, and node features with the class's corresponding minimum and maximum resource limits. Classes are searched in the order in which they are defined; when the first match is found, Moab assigns the job to that class.

You should not use remap classes to route jobs to queues/nodes in conjunction with a Torque routing queue. You should select only one of the two methods.

Because Moab remaps at job submission, updates you make to job requirements after submission will not cause any class changes. Moab does not restart the process.

In order to use `REMAPCLASS`, you must specify a `DEFAULTCLASS`. For example:

```
RMCFG[internal] DEFAULTCLASS=batch
```

In the example below, a job requesting 4 processors and the node feature `fast` are assigned to the class `quick`:

```
# You must specify a default class in order to use remap classes
RMCFG[internal]   DEFAULTCLASS=batch

# Jobs submitted to "batch" should be remapped
REMAPCLASS        batch

# stevens only queue
CLASSCFG[stevens] REQ.FEATURES=stevens REQUIREDUSERLIST=stevens,stevens2

# Special queue for I/O nodes
CLASSCFG[io]      MAX.PROC=8 REQ.FEATURES=io

# General access queues
CLASSCFG[quick]   MIN.PROC=2 MAX.PROC=8 REQ.FEATURES=fast|short
CLASSCFG[medium]  MIN.PROC=2 MAX.PROC=8
CLASSCFG[DEFAULT] MAX.PROC=64
...
```

The following parameters can be used to remap jobs to different classes:

- `MIN.PROC`
- `MAX.PROC`
- `MIN.TPN`
- `MAX.TPN`
- `MIN.WCLIMIT`
- `MAX.WCLIMIT`
- `REQ.FEATURES`
- `REQ.FLAGS=INTERACTIVE`
- `REQUIREDUSERLIST`

If the parameter REMAPCLASSLIST is set, then only the listed classes are searched and they are searched in the order specified by this parameter. If none of the listed classes are valid for a particular job, that job retains its original class.

The remap class only works with resource managers that allow dynamic modification of a job's assigned class/queue.

If default credentials are specified on a remap class, a job submitted to that class will inherit those credentials. If the destination class has different default credentials, the new defaults override the original settings. If the destination class does not have default credentials, the job maintains the defaults inherited from the remap class.

2.7.5.I Class Attribute Overview

The following table enumerates the different attributes for `CLASSCFG`.

Setting DEFAULT.* on a class does not assign resources or features to that class. Rather, it specifies resources that jobs will inherit when they are submitted to the class without their own resource requests. To configure features, use the parameter NODECFG.

CLASSCFG Parameters

DEFAULT.ATTR	
Format	<ATTRIBUTE>[,<ATTRIBUTE>]...
Description	One or more comma-delimited generic job attributes.
Example	---

DEFAULT.DISK	
Format	<INTEGER>
Description	Default amount of requested disk space.
Example	---

DEFAULT.EXT	
Format	<STRING>
Description	Default job resource manager extension.
Example	---

DEFAULT.FEATURESDEFAULT.EXT	
Format	Comma-delimited list of features.
Description	Default list of requested node features (a.k.a. node properties). This only applies to compute resource reqs.
Example	---

DEFAULT.GRES	
Format	<STRING>[<COUNT>][,<STRING>[<COUNT>]]...
Description	Default list of per task required consumable generic resources.

DEFAULT.GRES	
Example	`CLASSCFG[viz] DEFAULT.GRES=viz:2`

DEFAULT.MEM	
Format	<INTEGER> (in MB)
Description	Default amount of requested memory.
Example	---

DEFAULT.NODE	
Format	<INTEGER>
Description	Default required node count.
Example	`CLASSCFG[viz] DEFAULT.NODE=5` When a user submits a job to the `viz` class without a specified node count, the job is assigned 5 nodes.

DEFAULT.NODESET	
Format	<SETTYPE>:<SETATTR>[:<SETLIST>[,<SETLIST>]...]
Description	Default node set.
Example	`CLASSCFG[amd] DEFAULT.NODESET=ONEOF:FEATURE:ATHLON,OPTERON`

DEFAULT.PROC	
Format	<INTEGER>
Description	Default number of requested processors.
Example	---

DEFAULT.TPN	
Format	<INTEGER>
Description	Default number of tasks per node.
Example	---

DEFAULT.WCLIMIT	
Format	<INTEGER>
Description	Default wallclock limit.
Example	---

EXCL.FEATURES	
Format	Comma- or pipe-delimited list of node features.
Description	Set of excluded (disallowed) features. If delimited by commas, reject job if all features are requested; if delimited by the pipe symbol (|), reject job if at least one feature is requested.
Example	`CLASSCFG[intel] EXCL.FEATURES=ATHLON,AMD`

EXCL.FLAGS	
Format	Comma-delimited list of job flags.
Description	Set of excluded (disallowed) job flags. Reject job if any listed flags are set.
Example	`CLASSCFG[batch] EXCL.FLAGS=INTERACTIVE`

EXCLUDEUSERLIST	
Format	Comma-delimited list of users.
Description	List of users not permitted access to class.

EXCLUDEUSERLIST	
Example	---

FLAGS	
Format	NoBackfill
Description	Disable jobs from this class from backfilling.
Example	`CLASSCFG[batch] FLAGS=NoBackfill`

FORCENODEACCESSPOLICY	
Format	One of `SINGLETASK`, `SINGLEJOB`, `SINGLEUSER`, or `SHARED`
Description	Node access policy associated with queue. If set, this value overrides any per job settings specified by the user at the job level. See 4.3 Node Access Policies overview for more information.
Example	`CLASSCFG[batch] FORCENODEACCESSPOLICY=SINGLEJOB`

FSCAP	
Format	<DOUBLE>[%]
Description	See 5.3.1.A FSPOLICY - Specifying the Metric of Consumption specification.
Example	---

FSTARGET	
Format	<DOUBLE>[%]
Description	See 5.3.1.A FSPOLICY - Specifying the Metric of Consumption specification.
Example	---

HOSTLIST	
Format	Host expression, or comma-delimited list of hosts or host ranges.
Description	List of hosts associated with a class. If specified, Moab constrains the availability of a class to only nodes listed in the class host list.
Example	`CLASSCFG[batch] HOSTLIST=r:abs[45-113]`

IGNHOSTLIST	
Format	<BOOLEAN>
Default	FALSE
Description	If set to TRUE, any job submitted to the class will have its requested hostlist ignored by the scheduler.
Example	`CLASSCFG[batch] IGNHOSTLIST=TRUE`

JOBEPILOG	
Format	<STRING>
Description	Scheduler level job epilog to be run after job is completed by resource manager. See the section Special Class Attributes above.
Example	---

JOBFLAGS	
Format	Comma-delimited list of job flags.
Description	See 2.8 Job Flags for a description of flag values.
Example	`CLASSCFG[batch] JOBFLAGS=restartable`

JOBPROLOG	
Format	<STRING>
Description	Scheduler level job prolog to be run before job is started by resource manager. See the section Special Class Attributes above.
Example	---

JOBTRIGGER	
Format	<STRING>
Description	Job trigger associated with the class. See 17.3.1 Job Triggers.
Example	`CLASSCFG[batch] JOBTRIGGER=atype=exec,etype=create,action="/opt/moab/tools/job_trigger.pl"` *Execute `/opt/moab/tools/job_trigger.pl` when a job of class `batch` is created.*

MANAGERS	
Format	<USER>[,<USER>]...
Description	Users allowed to control, cancel, preempt, and modify jobs within class/queue. See the section Special Class Attributes above.
Example	`CLASSCFG[fast] MANAGERS=root,kerry,e43`

MAXJOB	
Format	<INTEGER>
Description	Maximum number of active (starting or running) jobs allowed in the class.
Example	---

MAXPROCPERNODE	
Format	<INTEGER>
Description	Maximum number of processors requested per node. May optionally include node names to articulate which nodes have a specific limit.

Example	`CLASSCFG[cpu] MAXPROCPERNODE=20  # When using this class, limit 20 for all nodes`
	`CLASSCFG[cpu] MAXPROCPERNODE[n1,n2]=20 MAXPROCPERNODE[n3]=10  # When using this class, limit 20 for n1 & n2 and limit 10 for n3`
	`CLASSCFG[cpu] MAXPROCPERNODE[n1,n2]=20 MAXPROCPERNODE=10 # When using this class, limit 20 for n1 & n2 and limit 10 for all other nodes`

MAX.CPUTIME	
Format	<INTEGER>
Description	Maximum allowed utilized CPU time.
Example	---

MAX.NODE	
Format	<INTEGER>
Description	Maximum number of requested nodes per job. Also used when the parameter REMAPCLASS is set to correctly route the job.
Example	`CLASSCFG[batch] MAX.NODE=64` Deny jobs requesting over 64 nodes access to the class `batch`.

MAX.PROC	
Format	<INTEGER>
Description	Maximum number of requested processors per job. Also used when the parameter REMAPCLASS is set to correctly route the job.

MAX.PROC	
	This enforces the requested processors, not the actual processors dedicated to a job. When enforcing limits for *NODEACCESSPOLICY SINGLEJOB*, use MAX.NODE instead.
Example	`CLASSCFG[small] MAX.PROC[USER]=3,6`

MAX.PS	
Format	<INTEGER>
Description	Maximum requested processor-seconds.
Example	---

MAX.TPN	
Format	<INTEGER>
Description	Maximum required tasks per node per job. Also used when the parameter REMAPCLASS is set to correctly route the job.
Example	---

MAX.WCLIMIT	
Format	[[[DD:]HH:]MM:]SS
Description	Maximum allowed wallclock limit per job. Also used when the parameter REMAPCLASS is set to correctly route the job.
Example	`CLASSCFG[long] MAX.WCLIMIT=96:00:00`

MIN.NODE	
Format	<INTEGER>

MIN.NODE	
Description	Minimum number of requested nodes per job. Also used when the parameter REMAPCLASS is set to correctly route the job.
Example	`CLASSCFG[dev] MIN.NODE=16` Jobs must request at least 16 nodes to be allowed to access the class.

MIN.PROC	
Format	<INTEGER>
Description	Minimum number of requested processors per job. Also used when the parameter REMAPCLASS is set to correctly route the job.
Example	`CLASSCFG[dev] MIN.PROC=32` Jobs must request at least 32 processors to be allowed to access the class.

MIN.PS	
Format	<INTEGER>
Description	Minimum requested processor-seconds.
Example	---

MIN.TPN	
Format	<INTEGER>
Description	Minimum required tasks per node per job. Also used when the parameter REMAPCLASS is set to correctly route the job.
Example	---

MIN.WCLIMIT	
Format	[[[DD:]HH:]MM:]SS

MIN.WCLIMIT	
Description	Minimum required wallclock limit per job. Also used when the parameter REMAPCLASS is set to correctly route the job.
Example	---

NODEACCESSPOLICY	
Format	One of `SINGLETASK`, `SINGLEJOB`, `SINGLEUSER`, or `SHARED`
Description	Default node access policy associated with queue. This value will be overridden by any per job settings specified by the user at the job level. See 4.3 Node Access Policies overview.
Example	`CLASSCFG[batch] NODEACCESSPOLICY=SINGLEJOB`

PARTITION	
Format	<STRING>
Description	Partition name where jobs associated with this class must run.
Example	`CLASSCFG[batch] PARTITION=p12`

PRIORITY	
Format	<INTEGER>
Description	Priority associated with the class. See 4.1.2 Job Priority Factors.
Example	`CLASSCFG[batch] PRIORITY=1000`

QDEF	
Format	<QOSID>
Description	Default QoS for jobs submitted to this class. You can specify a maximum of

QDEF	
	four QDEF entries per credential. Any QoSes specified after the fourth will not be accepted. In addition to classes, you can also specify QDEF for accounts, groups, and users.
Example	`CLASSCFG[batch] QDEF=base` Jobs submitted to class batch that do not explicitly request a QoS will have the QoS base assigned.

QLIST	
Format	<QOSID>[,<QOSID>]...
Description	List of accessible QoSes for jobs submitted to this class.
Example	`CLASSCFG[batch] QDEF=base QLIST=base,fast,special,bigio`

REQ.FEATURES	
Format	Comma- or pipe-delimited list of node features.
Description	Set of required features. If delimited by commas, all features are required; if delimited by the pipe symbol (|), at least one feature is required.
Example	`CLASSCFG[amd] REQ.FEATURES=ATHLON,AMD`

REQ.FLAGS	
Format	REQ.FLAGS can be used with only the INTERACTIVE flag.
Description	Sets the INTERACTIVE flag on jobs in this class.
Example	`CLASSCFG[orion] REQ.FLAGS=INTERACTIVE`

REQUIREDACCOUNTLIST	
Format	Comma-delimited list of accounts.
Description	List of accounts allowed to access and use a class (analogous to `*LIST` for other credentials).
Example	`CLASSCFG[jasper] REQUIREDACCOUNTLIST=testers,development`

REQUIREDUSERLIST	
Format	Comma-delimited list of users.
Description	List of users allowed to access and use a class (analogous to `*LIST` for other credentials).
Example	`CLASSCFG[jasper] REQUIREDUSERLIST=john,u13,steve,guest`

REQUIREDQOSLIST	
Format	Comma-delimited list of QoSes
Description	List of QoSes allowed to access and use a class (analogous to `*LIST` for other credentials). The number of unique QoSes is limited by the Moab Maximum ACL limit, which defaults to 32.
Example	`CLASSCFG[jasper] REQUIREDQOSLIST=hi,lo`

SYSPRIO	
Format	<INTEGER>
Description	Value of system priority applied to every job submitted to this class. Once a system priority has been added to a job, either manually or through configuration, it can only be removed manually.

SYSPRIO	
Example	`CLASSCFG[special] SYSPRIO=100`

WCOVERRUN	
Format	[[[DD:]HH:]MM:]SS
Description	Tolerated amount of time beyond the specified wallclock limit.
Example	---

2.7.5.J Enabling Queue Complex Functionality

Queue complexes allow you to build a hierarchy of queues and apply certain limits and rules to collections of these queues. Moab supports this functionality in two ways. The first way, queue mapping, is very simple but limited in functionality. The second method provides very rich functionality but requires more extensive configuration using the Moab hierarchical fairshare facility.

Queue Mapping

Queue mapping allows collections of queues to be mapped to a parent credential object against which various limits and policies can be applied, as in the following example:

```
QOSCFG[general]   MAXIJOB[USER]=14  PRIORITY=20
QOSCFG[prio]      MAXIJOB[USER]=8   PRIORITY=2000
# group short, med, and long jobs into 'general' QOS
CLASSCFG[short]   QDEF=general FSTARGET=30
CLASSCFG[med]     QDEF=general FSTARGET=40
CLASSCFG[long]    QDEF=general FSTARGET=30 MAXPROC=200
# group interactive and debug jobs into 'prio' QOS
CLASSCFG[inter]   QDEF=prio
CLASSCFG[debug]   QDEF=prio
CLASSCFG[premier] PRIORITY=10000
```

2.7.6 QoS Credential

The concept of a quality of service (QoS) credential is unique to Moab and is not derived from any underlying concept or peer service. In most cases, the QoS credential is used to set up a selection of service levels for end-users to choose from on a long-term or job-by-job basis. QoSes differ from other credentials in that they are centered around special access where this access may allow use of additional services, additional resources, or improved responsiveness. Unique to this credential, organizations can also choose to apply different charge rates to the varying levels of service available within each QoS. As QoS is an internal credential, all QoS configuration occurs within Moab.

QoS access and QoS defaults can be mapped to users, groups, accounts, and classes, allowing limited service offering for key users. As mentioned, these services focus around increasing access to special scheduling capabilities and additional resources and improving job responsiveness.

At a high level, unique QoS attributes can be broken down into the following:

- QoS Usage Limit Overrides
- QoS Service Targets
- QoS Privilege Flags
- QoS Charge Rate
- QoS Access Controls

2.7.6.A QoS Usage Limit Overrides

All credentials allow specification of job limits. In such cases, jobs are constrained by the most limiting of all applicable policies. With QoS override limits, however, jobs are limited by the override, regardless of other limits specified.

2.7.6.B QoS Service Targets

Service targets cause the scheduler to take certain job-related actions as various responsiveness targets are met. Targets can be set for either job queue time or job expansion factor and cause priority adjustments, reservation enforcement, or preemption activation. In strict service centric organizations, Moab can be configured to trigger various events and notifications in the case of failure by the cluster to meet responsiveness targets.

2.7.6.C QoS Privilege Flags

QoSes can provide access to special capabilities. These capabilities include preemption, job deadline support, backfill, next to run priority, guaranteed resource reservation, resource provisioning, dedicated resource access, and many others. See the complete list in 6.3 Quality of Service (QoS) Facilities.

2.7.6.D QoS Charge Rate

Associated with the QoSes many privileges is the ability to assign end-users costs for the use of these services. This charging can be done on a per-QoS basis and can be specified for both dedicated and use-based resource consumption. 6.3 Quality of Service (QoS) Facilities covers more details on QoS level costing configuration while Accounting, Charging, and Allocation Management provides more details regarding general single cluster and multi-cluster charging capabilities.

2.7.6.E QoS Access Controls

QoS access control can be enabled on a per QoS basis using the MEMBERULIST attribute or specified on a *per-requestor* basis using the `QDEF` and `QLIST` attributes of the USERCFG, GROUPCFG, ACCOUNTCFG, and CLASSCFG parameters. See 6.3.3 Managing QoS Access for more detail.

Related Topics

- 18.4 Identity Managers
- 5.2 Usage Limits/Throttling Policies

2.8 Job Flags

ADVRES	
Format	ADVRES[:<RESID>]
Default	Use available resources where ever found, whether inside a reservation or not.
Description	Specifies the job can only utilize accessible, reserved resources. If <RESID> is specified, only resources in the specified reservation can be utilized.
Example	`FLAGS=ADVRES:META.1` *The job can only utilize resources located in the `META.1` reservation.*

ALLPROCS	
Format	---
Default	---
Description	Each task should occupy all the processors on the node.

ALLPROCS	
	Incompatible with ppn and non-Torque systems. ALLPROCS is scheduled to be deprecated in a future Moab version where it will be replaced with the NUMA job submission syntax (place=node in this particular case).
Example	`msub -l nodes=6 -l flags=allprocs` *Each of the 6 tasks will occupy all the processors on the node and the job will launch enough processes to occupy each of those processors.*

ARRAYJOBPARLOCK	
Format	---
Default	---
Description	Specifies that the job array being submitted should not span across multiple partitions. This locks all subjobs of the array to a single partition. If you want to lock all job arrays to a single partition, specify the ARRAYJOBPARLOCK parameter in `moab.cfg` to force this behavior on a global scale.
Example	`> msub -t moab.[1-5]%3 -l walltime=30,flags=arrayjobparlock`

ARRAYJOBPARSPAN	
Format	---
Default	---
Description	Specifies that the job array being submitted should span across multiple partitions. This is the default behavior in Moab, unless the ARRAYJOBPARLOCK parameter is specified in `moab.cfg`. This job flag overrides the `ARRAYJOBPARLOCK` parameter so that job arrays can be allowed to span multiple partitions at submit time.
Example	`> msub -t moab.[1-5]%3 -l walltime=30,flags=arrayjobparspan`

FORCEPROVISION	
Format	FORCEPROVISION
Default	---
Description	A job will provision nodes whether or not they already have the requested OS. When provisioning is enabled (on KNL systems, for example) and this flag is present, the default provisioning behavior (where Moab does not provision a node if the current OS already matches the one being requested) is overridden.
Example	`msub -l os=RHEL,flags=forceprovision`

GRESONLY	
Format	GRESONLY
Default	False
Description	Uses no compute resources such as processors, memory, and so forth; uses only generic resources.
Example	`> msub -l gres=matlab,walltime=300`

IGNIDLEJOBRSV	
Format	IGNIDLEJOBRSV
Default	N/A
Description	Only applies to QOS. `IGNIDLEJOBRSV` allows jobs to start without a guaranteed walltime. Instead, it overlaps the idle reservations of real jobs and is preempted 2 minutes before the real job starts.
Example	`QOSCFG[standby] JOBFLAGS=IGNIDLEJOBRSV`

NOQUEUE	
Format	NOQUEUE

NOQUEUE	
Default	Jobs remain queued until they are able to run.
Description	Specifies that the job should be removed if it is unable to allocate resources and start execution immediately.
Example	`FLAGS=NOQUEUE` *The job should be removed unless it can start running at submit time.* This functionality is identical to the resource manager extension QUEUEJOB:FALSE.

NORMSTART	
Format	NORMSTART
Default	Moab passes jobs to a resource manager to schedule.
Description	Specifies that the job is an internal system job and will not be started via a resource manager.
Example	`FLAGS=NORMSTART` *The job begins running in Moab without a corresponding resource manager job.*

PREEMPTEE	
Format	PREEMPTEE
Default	Jobs cannot be preempted by other jobs
Description	Specifies that the job can be preempted by other jobs that have the `PREEMPTOR` flag set.
Example	`FLAGS=PREEMPTEE` *The job can be preempted by other jobs that have the `PREEMPTOR` flag set.*

PREEMPTOR	
Format	PREEMPTOR
Default	Jobs cannot preempt other jobs.
Description	Specifies that the job can preempt other jobs that have the `PREEMPTEE` flag set.
Example	`FLAGS=PREEMPTOR` *The job may preempt other jobs that have the `PREEMPTEE` flag set.*

PURGEONSUCCESSONLY	
Format	PURGEONSUCCESSONLY
Default	Completed jobs are sent to a queue for a short period of time before Moab purges them from the system.
Description	Specifies that Moab should only purge the job from the completed queue if it completed successfully. If the job failed, Moab will keep it in the queue indefinitely to allow you to restart it at any time. This flag is particularly useful for setup and take down jobs in job workflows. See 22.1.5 Creating Workflows with Job Templates for more information.
Example	`FLAGS=PURGEONSUCCESSONLY` *If the job fails, Moab will not purge it from the completed job queue.*

RESTARTABLE	
Format	RESTARTABLE
Default	Jobs cannot be restarted if preempted.
Description	Specifies jobs can be *requeued* and later restarted if preempted.
Example	`FLAGS=RESTARTABLE` *The associated job can be preempted and restarted at a later date.*

SUSPENDABLE	
Format	SUSPENDABLE
Default	Jobs cannot be suspended if preempted.
Description	Specifies jobs can be *suspended* and later resumed if preempted.
Example	`FLAGS=SUSPENDABLE` *The associated job can be suspended and resumed at a later date.*

SYSTEMJOB	
Format	SYSTEMJOB
Default	N/A
Description	Creates an internal system job that does not require resources.
Example	`FLAGS=SYSTEMJOB`

USEMOABJOBID	
Format	<BOOLEAN>
Default	FALSE
Description	Specifies whether to return the Moab job ID when running 'msub', or the resource manager's job ID if it is available. Setting USEMOABJOBID here overrides the global setting for USEMOABJOBID in moabcfg. See the USEMOABJOBID parameter for more information.
Example	`FLAGS=USEMOABJOBID SELECT=TRUE`

WIDERSVSEARCHALGO	
Format	<BOOLEAN>
Default	---
Description	When Moab is determining when and where a job can run, it either searches for the most resources or the longest range of resources. In almost all cases searching for the longest range is ideal and returns the soonest starttime. In some rare cases, however, a particular job may need to search for the most resources. In those cases this flag can be used to have the job find the soonest starttime. The flag can be specified at submit time, or you can use mjobctl -m to modify the job after it has been submitted. See the RSVSEARCHALGO parameter.

Example	`> msub -l flags=widersvsearchalgo`
	`> mjobctl -m flags+=widersvsearchalgo job.1`

Related Topics

- JOBFLAGS

Chapter 3: Scheduler Commands

In this chapter:

3.1 Moab Command Overview

In this section:

3.1.1 Moab Commands

Command	Description
checkjob	Provide detailed status report for specified job
checknode	Provide detailed status report for specified node
mcredctl	Controls various aspects about the credential objects within Moab
mdiag	Provide diagnostic reports for resources, workload, and scheduling
mjobctl	Control and modify job

Command	Description
mnodectl	Control and modify nodes
moab	Control the Moab daemon
mrmctl	Query and control resource managers
mrsvctl	Create, control and modify reservations
mschedctl	Modify scheduler state and behavior
mshow	Displays various diagnostic messages about the system and job queues
mshow -a	Query and show available system resources
msub	Scheduler job submission
mvcctl	Create, modify, and delete VCs
showbf	Show current resource availability
showhist.moab.pl	Show past job information
showq	Show queued jobs
showres	Show existing reservations
showstart	Show estimates of when job can/will start
showstate	Show current state of resources
showstats	Show usage statistics
showstats -f	Show various tables of scheduling/system performance

3.1.2 Moab Command Options

For many Moab commands, you can use the following options to specify that Moab will run the command in a different way or different location from the configured default. These options do not change your settings in the configuration file; they override the settings for this single instance of the command.

Option	Description
--about	Displays build and version information and the status of your Moab license.
--help	Displays usage information about the command.
--host=<serverHostName>	Causes Moab to run the client command on the specified host.
--loglevel=<logLevel>	Causes Moab to write log information to STDERR as the client command is running. For more information, see 12.2 Logging.
--msg=<message>	Causes Moab to annotate the action in the event log.
--port=<serverPort>	Causes Moab to run the command using the port specified.
--timeout=<seconds>	Sets the maximum time that the client command will wait for a response from the Moab server.
--version	Displays version information.
--xml	Causes Moab to return the command output in XML format.

3.1.3 Commands Providing Maui Compatibility

The following commands are deprecated. Click the link for respective deprecated commands to see the updated replacement command for each.

Command	Description
canceljob	Cancel job
changeparam	Change in memory parameter settings.
diagnose	Provide diagnostic report for various aspects of resources, workload, and scheduling.

Command	Description
releasehold	Release job defers and holds.
releaseres	Release reservations.
runjob	Force a job to run immediately.
sethold	Set job holds.
setqos	Modify job QOS settings.
setres	Set an admin/user reservation.
setspri	Adjust job/system priority of job.
showconfig	Show current scheduler configuration.

3.2 Status Commands

The status commands organize and present information about the current state and historical statistics of the scheduler, jobs, resources, users, and accounts. The following table presents the primary status commands and flags:

Command	Description
checkjob	Displays detailed job information such as job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.
checknode	Displays detailed node information such as node state, resources, attributes, reservations, history, and statistics.
mdiag -f	Displays summarized fairshare information and any unexpected fairshare configuration.
mdiag -j	Displays summarized job information and any unexpected job state.
mdiag -n	Displays summarized node information and any unexpected node state.
mdiag -p	Displays summarized job priority information.

Command	Description
mschedctl -f	Resets internal statistics.
showstats -f	Displays various aspects of scheduling performance across a job duration/job size matrix.
showq [-r|-i]	Displays various views of currently queued active, idle, and non-eligible jobs.
showstats -g	Displays current and historical usage on a per group basis.
showstats -u	Displays current and historical usage on a per user basis.
showstats -v	Displays high level current and historical scheduling statistics.

3.3 Job Management Commands

Moab shares job management tasks with the resource manager. Typically, the scheduler only modifies scheduling relevant aspects of the job such as partition access, job priority, charge account, and hold state. The following table covers the available job management commands. Section 3.1 Moab Command Overview lists all available commands.

Command	Description
canceljob	Cancels existing job.
checkjob	Displays job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.
mdiag -j	Displays summarized job information and any unexpected job state.
releasehold -a	Removes job holds or deferrals.
runjob	Starts job immediately, if possible.
sethold	Sets hold on job.
setqos	Sets/modifies QoS of existing job.
setspri	Adjusts job/system priority of job.

Related Topics

- 2.3.1.A Job States

3.4 Reservation Management Commands

Moab exclusively controls and manages all advance reservation features including both standing and administrative reservations. The following table covers the available reservation management commands:

Command	Description
mdiag -r	Displays summarized reservation information and any unexpected state.
mrsvctl	Reservation control.
mrsvctl -r	Removes reservations.
mrsvctl -c	Creates an administrative reservation.
showres	Displays information regarding location and state of reservations.

3.5 Policy/Configuration Management Commands

Moab allows dynamic modification of most scheduling parameters allowing new scheduling policies, algorithms, constraints, and permissions to be set at any time. Changes made via Moab client commands are temporary and are overridden by values specified in Moab configuration files the next time Moab is shut down and restarted. The following table covers the available configuration management commands:

Command	Description
mschedctl -l	Displays triggers, messages, and settings of all configuration parameters.
mschedctl	Controls the scheduler (behavior, parameters, triggers, messages).
mschedctl -m	Modifies system values.

3.6 End-User Commands

While the majority of Moab commands are tailored for use by system admins, a number of commands are designed to extend the knowledge and capabilities of end-users. The following table covers the commands available to end-users.

When using Active Directory as a central authentication mechanism, all nodes must be reported with a different name when booted in both Linux and Windows (for instance, `node01-l` for Linux and `node01` for Windows). If a machine account with the same name is created for each OS, the most recent OS will remove the previously-joined machine account. The nodes must report to Moab with the same hostname. This can be done by using aliases (adding all node names to the `/etc/hosts` file on the system where Moab is running) and ensuring that the Linux resource manager reports the node with its global name rather than the Linux-specific one (`node01` rather than `node01-l`).

Command	Description
canceljob	Cancels existing job.
checkjob	Displays job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization.
msub	Submit a new job.
releaseres	Releases a user reservation.
setres	Create a user reservation.
showbf	Shows resource availability for jobs with specific resource requirements.
showq	Displays detailed prioritized list of active and idle jobs.
showstart	Shows estimated start time of idle jobs.
showstats	Shows detailed usage statistics for users, groups, and accounts, to which the end-user has access.

Related Topics

- Chapter 3: Scheduler Commands

3.7 Moab Commands

See the Moab commands below.

checkjob
checknode
mcredctl
mdiag
mjobctl
mnodectl
moab
mrmctl
mrsvctl
mschedctl
mshow
mshow -a
msub
mvcctl
showbf
showhist.moab.pl
showq
showres
showstart

showstate
showstats
showstats -f

3.7.1 checkjob

3.7.1.A Synopsis

`checkjob` [exact:*jobid*] [jobname:*jobname*] [-l *policylevel*] [-n *nodeid*] [-q *qosid*] [-r *reservationid*] [-v] [--flags=future | complete] [--blocking] *jobid* `[--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]`

3.7.1.B Overview

`checkjob` displays detailed job state information and diagnostic output for a specified job. Detailed information is available for queued, blocked, active, and recently completed jobs. The checkjob command shows the master job of an array, and a summary of array subjobs but does not display all subjobs. Use checkjob -v to display all job-array subjobs.

3.7.1.C Access

This command can be run by level 1-3 Moab Admins for any job. Also, end users can use `checkjob` to view the status of their own jobs.

3.7.1.D Options

--blocking	
Format	`--blocking`
Description	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
Example	`> checkjob -v --blocking 1234`

--blocking	
	Display real time data about job 1234.

--flags	
Format	`--flags=future | complete`
Description	• `future` – Evaluates future eligibility of job (ignore current resource state and usage limitations). • `complete` – Queries details for jobs that have already terminated.
Example	`> checkjob -v --flags=future 6235` *Display reasons why idle job is blocked ignoring node state and current node utilization constraints.*

exact	
Format	exact:`<JOBID>`
Description	Searches for and returns the exact job ID.
Example	`> checkjob exact:1.job_dependency1`

jobname	
Format	`jobname:<JOBNAME>`
Description	Searches for and returns the first job with the matching `JOBNAME`.
Example	`> checkjob jobname:STEP4`

-l (Policy level)	

Format	`<POLICYLEVEL>`
	
	`HARD`, `SOFT`, or `OFF`

Description	Reports job start eligibility subject to specified throttling policy level.

-l (Policy level)	

Example	`> checkjob -l SOFT 6235`
	`> checkjob -l HARD 6235`

-n (NodeID)	
Format	`<NODEID>`
Description	Checks job access to specified node and preemption status with regards to jobs located on that node.
Example	`> checkjob -n node113 6235`

-q (QoS)	
Format	`<QOSID>`
Description	Checks job access to specified QoS `<QOSID>`.
Example	`> checkjob -q special 6235`

-r (Reservation)	
Format	`<RSVID>`
Description	Checks job access to specified reservation `<RSVID>`.
Example	`> checkjob -r orion.1 6235`

-v (Verbose)	
Description	Sets verbose mode. If the job is part of an array, the `-v` option shows pertinent array information before the job-specific information (see Example 2 and Example 3 for differences between standard output and -v output). Specifying the double verbose (`-v -v`) displays additional information about the job. See the Output table below for details.
Example	`> checkjob -v 6235`

3.7.1.E Details

This command allows any Moab Admin to check the detailed status and resource requirements of an active, queued, or recently completed job. Additionally, this command performs numerous diagnostic checks and determines if and where the job could potentially run. Diagnostic checks include policy violations, reservation constraints, preemption status, and job to resource mapping. If a job cannot run, a text reason is provided along with a summary of how many nodes are and are not available. If the `-v` flag is specified, a node by node summary of resource availability will be displayed for idle jobs.

Job Eligibility

If a job cannot run, a text reason is provided along with a summary of how many nodes are and are not available. If the `-v` flag is specified, a node by node summary of resource availability will be displayed for idle jobs. For job level eligibility issues, one of the following reasons will be given:

Reason	Description
Job has hold in place	One or more job holds are currently in place
Insufficient idle procs	There are currently not adequate processor resources available to start the job
Idle procs do not meet requirements	adequate idle processors are available but these do not meet job requirements
Start date not reached	Job has specified a minimum *start date* that is still in the future
Expected state is not idle	Job is in an unexpected state
State is not idle	Job is not in the idle state
Dependency is not met	Job depends on another job reaching a certain state
Rejected by policy	Job start is prevented by a throttling policy

If a job cannot run on a particular node, one of the following 'per node' reasons will be given:

Reason	Description
Class	Node does not allow required job class/queue
CPU	Node does not possess required processors
Disk	Node does not possess required local disk
Features	Node does not possess required node features
Memory	Node does not possess required real memory
Network	Node does not possess required network interface
State	Node is not Idle or Running

Reservation Access

The -r flag can be used to provide detailed information about job access to a specific reservation.

Preemption Status

If a job is marked as a preemptor and the -v and -n flags are specified, `checkjob` will perform a job by job analysis for all jobs on the specified node to determine if they can be preempted.

3.7.1.F Output

The `checkjob` command displays the following job attributes:

Attribute	Value	Description
Account	`<STRING>`	Name of account associated with job.
Allocated Nodes	Square bracket delimited list of node and processor IDs	List of nodes and processors allocated to job.
Applied Nodeset**	`<STRING>`	Nodeset used for the job's node allocation.
Arch	`<STRING>`	Node architecture

Attribute	Value	Description
		required by job.
Attr	square bracket delimited list of job attributes	Job Attributes (i.e., `[BACKFILL]` `[PREEMPTEE]`).
Available Memory**	`<INTEGER>`	The available memory requested by job. Moab displays the relative or exact value by returning a comparison symbol (>, <, >=, <=, or ==) with the value (i.e., `Available Memory <= 2048`).
Available Swap**	`<INTEGER>`	The available swap requested by job. Moab displays the relative or exact value by returning a comparison symbol (>, <, >=, <=, or ==) with the value (i.e., `Available Swap >= 1024`).
Average Utilized Procs*	`<FLOAT>`	Average load balance for a job.
Avg Util Resources Per Task*	`<FLOAT>`	
BecameEligible	`<TIMESTAMP>`	The date and time when the job moved from Blocked to Eligible.
Bypass	`<INTEGER>`	Number of times a lower priority job with a later submit time ran before the job.
CheckpointStartTime**	`[[[DD:]HH:]MM:]SS`	The time the job was first checkpointed.
Class	[`<CLASS NAME>` `<CLASS COUNT>`]	Name of class/queue required by job and number of class initiators

Attribute	Value	Description
		required per task.
Dedicated Resources Per Task*	Space-delimited list of `<STRING>` :`<INTEGER>`	Resources dedicated to a job on a per-task basis.
Disk	`<INTEGER>`	Amount of local disk required by job (in MB).
Estimated Walltime	[[[DD:]HH:]MM:]SS	The scheduler's estimated walltime.
EnvVariables**	Comma-delimited list of `<STRING>`	List of environment variables assigned to job.
Exec Size*	`<INTEGER>`	Size of job executable (in MB).
Executable	`<STRING>`	Name of command to run.
Features	Square bracket delimited list of `<STRING>`s	Node features required by job.
Flags	Comma-delimited list of `<STRING>` job flags	List of job flags.
Group	`<STRING>`	Name of UNIX group associated with job.
Holds	Zero or more of User, System, and Batch	Types of job holds currently applied to job.
Image Size	`<INTEGER>`	Size of job data (in MB).
IWD (Initial Working Directory)	`<DIR>`	Directory to run the executable in.
Job Messages**	`<STRING>`	Messages attached to a job.
Job Submission**	`<STRING>`	Job script submitted to

Attribute	Value	Description
		resource manager.
Memory	`<INTEGER>`	Amount of real memory required per node (in MB).
Max Util Resources Per Task*	`<FLOAT>`	
NodeAccess*	`<STRING>`	Node access policy associated with job.
Nodecount	`<INTEGER>`	Number of nodes required by job.
Opsys	`<STRING>`	Node operating system required by job.
Partition Mask	ALL or colon-delimited list of partitions	List of partitions the job has access to.
PE	`<FLOAT>`	Number of processor-equivalents requested by job.
Per Partition Priority**	Tabular	Table showing job template priority for each partition.
Priority Analysis**	Tabular	Table showing how the job's priority was calculated: `Job PRIORITY* Cred (User:Group:Class) Serv(QTime)`
QOS	`<STRING>`	Quality of Service associated with job.
Reservation	`<RSVID>` (`<TIME1>` - `<TIME2>` Duration: `<TIME3>`)	RESID specifies the reservation ID, TIME1 is the relative start time, TIME2 the relative end time, TIME3 the duration of the reservation.

Attribute	Value	Description
Req	[<INTEGER>] TaskCount: <INTEGER> Partition: <partition>	A job requirement for a single type of resource followed by the number of tasks instances required and the appropriate partition.
StageIn	<SOURCE> %<DESTINATION>	The <SOURCE> is the username, hostname, directory and file name of origin for the file(s) that Moab will stage in for this job. The <DESTINATION> is the username, hostname, directory and file name where Moab will place the file during this job. See Chapter 24: Data Staging for more information.
StageInSize	<INTEGER><UNIT>	The size of the file Moab will stage in for this job. <UNIT> can be KB, MB, GB, or TB. See Chapter 24: Data Staging for more information.
StageOut	<SOURCE> %<DESTINATION>	The <SOURCE> is the username, hostname, directory and file name of origin for the file(s) that Moab will stage out for this job. The <DESTINATION> is the username, hostname, directory and file name where Moab will place the file during this job. See Chapter 24: Data Staging for more information.
StageOutSize	<INTEGER><UNIT>	The size of the file Moab

Attribute	Value	Description
		will stage out for this job. `<UNIT>` can be KB, MB, GB, or TB. See Chapter 24: Data Staging for more information.
StartCount	`<INTEGER>`	Number of times job has been started by Moab.
StartPriority	`<INTEGER>`	Start priority of job.
StartTime	`<TIME>`	Time job was started by the resource management system.
State	One of Idle, Starting, Running, etc. See 2.3.1.A Job States for all possible values.	Current Job State.
SubmitTime	`<TIME>`	Time job was submitted to resource management system.
Swap	`<INTEGER>`	Amount of swap disk required by job (in MB).
Task Distribution*	Square bracket delimited list of nodes	
Time Queued	Total: [[[DD:]HH:]MM:]SS Eligible: [[[DD:]HH:]MM:]SS	Total duration of time the job was queued and duration of time that the job was eligible to run.
Total Requested Nodes**	`<INTEGER>`	Number of nodes the job requested.
Total Requested Tasks	`<INTEGER>`	Number of tasks requested by job.
User	`<STRING>`	Name of user submitting job.

Attribute	Value	Description
Utilized Resources Per Task*	`<FLOAT>`	
WallTime	[[[DD:]HH:]MM:]SS of [[[DD:]HH:]MM:]SS	Length of time job has been running out of the specified limit.

In the above table, fields marked with an asterisk (*) are only displayed when set or when the `-v` flag is specified. Fields marked with two asterisks (**) are only displayed when set or when the `-v -v` flag is specified.

Example 3-1: checkjob 717

```
> checkjob 717
job 717
State: Idle
Creds:  user:jacksond  group:jacksond  class:batch
WallTime: 00:00:00 of 00:01:40
SubmitTime: Mon Aug 15 20:49:41
  (Time Queued  Total: 3:12:23:13  Eligible: 3:12:23:11)
TerminationDate:   INFINITY  Sat Oct 24 06:26:40
Total Tasks: 1
Req[0]  TaskCount: 1  Partition: ALL
Network: ---  Memory >= 0  Disk >= 0  Swap >= 0
Opsys: ---  Arch: ---  Features: ---

IWD:            /home/jacksond/moab/moab-x.x.x
Executable:     STDIN
Flags:          RESTARTABLE,NORMSTART
StartPriority:  5063
Reservation '717' (  INFINITY ->   INFINITY  Duration: 00:01:40)
Note:  job cannot run in partition base (idle procs do not meet requirements : 0 of 1
procs found)
idle procs:   4  feasible procs:   0
Rejection Reasons: [State        :    3][ReserveTime  :    1]
cannot select job 717 for partition GM (partition GM does not support requested class
batch)
```

The example job cannot be started for two different reasons:

- It is temporarily blocked from partition `base` because of node state and node reservation conflicts.
- It is permanently blocked from partition `GM` because the requested class `batch` is not supported in that partition.

Example 3-2: Using `checkjob` (no -v) on a job array master job

```
checkjob array.1
job array.1

AName: array
Job Array Info:
  Name: array.1

Sub-jobs:         10
  Active:          6 ( 60.0%)
  Eligible:        2 ( 20.0%)
  Blocked:         2 ( 20.0%)
  Complete:        0 (  0.0%)
```

Example 3-3: Using `checkjob -v` on a job array master job

```
$ checkjob -v array.1
job array.1

AName: array
Job Array Info:
  Name: array.1
  1 : array.1.1 : Running
  2 : array.1.2 : Running
  3 : array.1.3 : Running
  4 : array.1.4 : Running
  5 : array.1.5 : Running
  6 : array.1.6 : Running
  7 : array.1.7 : Idle
  8 : array.1.8 : Idle
  9 : array.1.9 : Blocked
  10 : array.1.10 : Blocked

Sub-jobs:         10
  Active:          6 ( 60.0%)
  Eligible:        2 ( 20.0%)
  Blocked:         2 ( 20.0%)
  Complete:        0 (  0.0%)
```

Example 3-4: Using checkjob -v on a data staging job

```
$ checkjob -v moab.14.dsin
job moab.14.dsin

AName: moab.14.dsin
State: Running
Creds:  user:fred  group:company
WallTime:   00:00:00 of 00:01:01
SubmitTime: Wed Apr 16 10:07:19
 (Time Queued  Total: 00:00:00  Eligible: 00:00:00)

StartTime: Wed Apr 16 10:07:19
TemplateSets:  dsin
Triggers: 78$start+0@0.000000:exec@/opt/moab/tools/datastaging/ds_move_rsync --
stagein:FALSE
Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: SHARED
Dedicated Resources Per Task: bandwidth: 1
NodeAccess: SHARED

Allocated Nodes:
[GLOBAL:1]

Job Group:  moab.14
SystemID:   moab
SystemJID:  moab.14.dsin
Task Distribution: GLOBAL
```

```
IWD:            $HOME/test/datastaging
SubmitDir:      $HOME/test/datastaging
StartCount:     1
Parent VCs:     vc11
User Specified Partition List:   local
Partition List: local
SrcRM:          internal
Flags:          NORMSTART,GRESONLY,TEMPLATESAPPLIED
Attr:           dsin
StageInSize:    386MB
StageOutSize:   100MB
StageIn:        fred@remotelab:/home/fred/input1/%fred@scratch:/home/fred/input1/
StageIn:        fred@remotelab:/home/fred/input2/%fred@scratch:/home/fred/input2/
StageIn:        fred@remotelab:/home/fred/input3/%fred@scratch:/home/fred/input3/
StageOut:       fred@scratch:/home/fred/output/%fred@remotelab:/home/fred/output/
StartPriority:  1
  SJob Type:             datastaging
  Completion Policy:     datastaging
PE:             0.00
Reservation 'moab.14.dsin' (-00:00:06 -> 00:00:55  Duration: 00:01:01)
```

Related Topics

- 3.7.32 showhist.moab.pl - explains how to query for past job information
- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.9 mdiag -j - command that displays additional detailed information regarding jobs
- 3.7.31 showq - command that shows high-level job summaries
- JOBCPURGETIME parameter - specify how long information regarding completed jobs is maintained
- 21.2.5 Testing and Troubleshooting Preemption

3.7.2 checknode

3.7.2.A Synopsis

`checknode` *flags* [*nodeID* | *ALL*] `[--about]` `[--help]` `[--host=<serverHostName>]` `[--loglevel=<logLevel>]` `[--msg=<message>]` `[--port=<serverPort>]` `[--timeout=<seconds>]` `[--version]` `[--xml]`

3.7.2.B Overview

This command shows detailed state information and statistics for nodes that run jobs.

The following information is returned by this command:

Name	Description
ACL	Node Access Control List (displayed only if set).
ActiveTime	Total time node has been busy (allocated to active jobs) since statistics initialization expressed in `HH:MM:SS` notation (percent of time busy: BusyTime/TotalTime).
Adapters	Network adapters available.
Arch	Architecture.
Classes	Classes available.
Disk	Disk space available.
Downtime	Displayed only if downtime is scheduled.
EffNodeAccessPolicy	Configured effective node access policy.
Features	Features available.
Load	CPU Load (Berkley one-minute load average).
Memory	Memory available.
Opsys	Operating system.
RequestID	Dynamic Node RequestID set by the resource manager (displayed only if set).
State	Node state.
StateTime	Time node has been in current state in HH:MM:SS notation.
Swap	Swap space available.
TotalTime	Total time node has been detected since statistics initialization expressed in `HH:MM:SS` notation.
TTL	Dynamic Node Time To Live set by the resource manager (expiration date, displayed only if set).

Name	Description
UpTime	Total time node has been in an available (Non-Down) state since statistics initialization expressed in `HH:MM:SS` notation (percent of time up: UpTime/TotalTime).

After displaying this information, some analysis is performed and any unusual conditions are reported.

3.7.2.C Access

By default, this command can be run by any Moab Admin (see the parameter ADMINCFG).

3.7.2.D Parameters

Name	Description
NODE	Node name you want to check. Moab uses regular expressions to return any node that contains the provided argument. For example, if you ran *`checknode node1`*, Moab returns information about `node1`, `node10`, `node100`, etc. If you want to limit the results to `node1` only, you run *`checknode "^node1$"`*.

3.7.2.E Flags

Name	Description
ALL	Returns checknode output on all nodes in the cluster.
-h	Help for this command.
-v	Returns verbose output.
--xml	Output in XML format. Same as mdiag -n --xml.

Example 3-5: checknode

```
> checknode P690-032
node P690-032

State:      Busy  (in current state for 11:31:10)
Configured Resources: PROCS: 1  MEM: 16G  SWAP: 2000M  DISK: 500G
Utilized   Resources: PROCS: 1
Dedicated  Resources: PROCS: 1
```

```
Opsys:      AIX       Arch:      P690
Speed:      1.00      CPULoad:   1.000
Network:    InfiniBand,Myrinet
Features:   Myrinet
Attributes: [Batch]
Classes:    [batch]

Total Time: 5:23:28:36  Up: 5:23:28:36 (100.00%)  Active: 5:19:44:22 (97.40%)

Reservations:
  Job '13678'(x1)  10:16:12:22 -> 12:16:12:22 (2:00:00:00)
  Job '13186'(x1)  -11:31:10 -> 1:12:28:50 (2:00:00:00)
Jobs:  13186
```

Example 3-6: checknode ALL

```
> checknode ALL
node ahe

State:      Idle  (in current state for 00:00:30)
Configured Resources: PROCS: 12  MEM: 8004M  SWAP: 26G  DISK: 1M
Utilized   Resources: PROCS: 1  SWAP: 4106M
Dedicated  Resources: ---
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   1.400
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:00 (0.00%)

Reservations:  ---
node ahe-ubuntu32

State:   Running  (in current state for 00:00:05)
Configured Resources: PROCS: 12  MEM: 2013M  SWAP: 3405M  DISK: 1M
Utilized   Resources: PROCS: 6  SWAP: 55M
Dedicated  Resources: PROCS: 6
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   2.000
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:02 (1.92%)

Reservations:
  6x2  Job:Running  -00:00:07 -> 00:01:53 (00:02:00)
  7x2  Job:Running  -00:00:06 -> 00:01:54 (00:02:00)
  8x2  Job:Running  -00:00:05 -> 00:01:55 (00:02:00)
Jobs:        6,7,8
node ahe-ubuntu64

State:      Busy  (in current state for 00:00:06)
Configured Resources: PROCS: 12  MEM: 2008M  SWAP: 3317M  DISK: 1M
Utilized   Resources: PROCS: 12  SWAP: 359M
Dedicated  Resources: PROCS: 12
```

```
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.000
Flags:      rmdetected
Classes:    [batch]
RM[ahe]*:   TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 00:01:44  Up: 00:01:44 (100.00%)  Active: 00:00:55 (52.88%)

Reservations:
  0x2  Job:Running  -00:01:10 -> 00:00:50 (00:02:00)
  1x2  Job:Running  -00:00:20 -> 00:01:40 (00:02:00)
  2x2  Job:Running  -00:00:20 -> 00:01:40 (00:02:00)
  3x2  Job:Running  -00:00:17 -> 00:01:43 (00:02:00)
  4x2  Job:Running  -00:00:13 -> 00:01:47 (00:02:00)
  5x2  Job:Running  -00:00:07 -> 00:01:53 (00:02:00)
Jobs:        0,1,2,3,4,5
ALERT:  node is in state Busy but load is low (0.000)
```

Example 3-7: checknode n001 (Dynamic Node)

```
> checknode node001
node node001

State:      Idle  (in current state for 00:13:50)
Configured Resources: PROCS: 2  MEM: 4096M
Utilized   Resources: PROCS: 2
Dedicated  Resources: ---
ACL:        USER==FRED+:==BOB+ GROUP==DEV+
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      ---       Arch:      ---
Speed:      1.00      CPULoad:   2.000
Partition:  local  Rack/Slot:  ---  NodeIndex:  1
RM[local]*: TYPE=NATIVE:AGFULL
EffNodeAccessPolicy: SHARED
RequestID:  1234
TTL:  Tue Nov 10 00:00:00 2024
Total Time: 2:21:19:05  Up: 2:21:19:05 (100.00%)  Active: 00:00:00 (0.00%)

Reservations:
  node001-TTL-1234x1  User     441days ->   INFINITY (  INFINITY)
    Blocked Resources@   441days  Procs: 2/2 (100.00%)  Mem: 4096/4096 (100.00%)
Swap: 1/1 (100.00%)  Disk: 1/1 (100.00%)
ALERT:  node is in state Idle but load is high (2.000)
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.10 mdiag -n
- 3.7.35 showstate

3.7.3 mcredctl

3.7.3.A Synopsis

mcredctl [-d credtype[:credid]] [-h credtype:credid] [-l credtype] [-q {role|limit|profile|accessfrom|accessto|policies} credtype[:credid]] [--format=xml] [-r {stats|fairshare|uid} <type>[:<ID>] [-t <STARTTIME>[,<ENDTIME>] [--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]

3.7.3.B Overview

The *mcredctl* command controls various aspects about the credential objects within Moab. It can be used to display configuration, limits, roles, and relationships for various Moab credential objects.

If using Insight, you must restart Moab to view credential modifications.

3.7.3.C Options

In all cases <CREDTYPE> is one of acct, group, user, class, or qos.

In most cases it is necessary to use the --format=xml flag in order to print the output (see examples below for specific syntax requirements).

-d - DESTROY	
Format	<TYPE>:<VAL>
Description	Purge a credential from moab.cfg (does not delete credential from memory).
Example	`> mcredctl -d user:john` *All references to USERCFG[john] will be commented out of moab.cfg)*

-h - HOLD	
Format	<TYPE>:<VAL>

-h - HOLD	
Description	Toggles whether a given credential's jobs should be placed on hold or not.
Example	`> mcredctl -h user:john` *User [john] will be put on hold.*

-l - LIST	
Format	`<TYPE>`
Description	List the various sub-objects of the specified credential.

Example	`> mcredctl -l user --format=xml`
	List all users within Moab in XML.
	`> mcredctl -l group --format=xml`
	List all groups within Moab in XML.

-q - QUERY	
Format	{role | accessfrom | accessto | limit| profile | policies} limit `<TYPE>` policies `<TYPE>` role `<USER>:<USERID>` profile `<TYPE>[:<VAL>]` accessfrom `<TYPE>[:<VAL>]` accessto `<TYPE>[:<VAL>]`
Description	Display various aspects of a credential (formatted in XML).

Example	`> mcredctl -q role user:bob --format=xml`
	View user `bob`'s administrative role within Moab in XML.
	`> mcredctl -q limit acct --format=xml`
	Display limits for all accounts in XML.
	`> mcredctl -q policies user:bob`

-q - QUERY	
	View limits organized by credential for user `bob` on each partition and resource manager. `> mcredctl -q profile group --format=xml --timeout=00:10:00 -o time:1388590200,1431529200,types:TPSD` *Generates a report of processor hours used by groups per month. `TPSD` represents total proc-seconds dedicated by this credential in the profiling interval.*

-r - RESET	
Format	{stats|fairshare|uid} `<TYPE>` [:`<ID>`]
Description	Reset the stats, fairshare, or UID/GID of a given credential. When resetting UID, only a type of user is supported.
Example	`> mcredctl -r uid user:john` *Resets the UID/GID for the user named john.*

-t - TIMEFRAME	
Format	`<STARTTIME>`[,`<ENDTIME>`]
Description	Can be used in conjunction with the `-q` profile option to display profiling information for the specified timeframe.
Example	`> mcredctl -q profile user -t 14:30_06/23`

3.7.3.D Credential Statistics XML Output

Credential statistics can be requested as XML (via the `--format=xml` argument) and will be written to STDOUT in the following format:

```
> mcredctl -q profile user --format=xml -o time:1182927600,1183013999
<Data>
  <user ...>
    <Profile ...>
    </Profile>
  </user>
</Data>
```

Example 3-8: Deleting a group

```
> mcredctl -d group:john
GROUPCFG[john] Successfully purged from config files
```

Example 3-9: List users in XML format

```
> mcredctl -l user --format=xml
<Data><user ID="john"</user><user ID="john"></user><user ID="root"></user><user
ID="dev"></user></Data>
```

Example 3-10: Display information about a user

```
> mcredctl -q role user:john --format=xml
<Data><user ID="test" role="admin5"></user></Data>
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes

3.7.4 mdiag

3.7.4.A Synopsis

```
mdiag -a [accountid]

mdiag -b [-l policylevel] [-t partition] [-v]

mdiag -c [classid] [-v]

mdiag -C [configfile]

mdiag -f [-o user|group|acct|qos|class] [-v] [--
flags=relative]

mdiag -g [groupid]

mdiag -G [Green]

mdiag -j [jobid] [-t <partition>] [-v][-w
state|user|account|class|group|qos=VALUE][--flags=policy][--
blocking]

mdiag -l

mdiag -L [-v]
mdiag -n [-A <creds>] [-t partition] [nodeid] [-v]
mdiag -p [-t partition] [-v] [-v]
mdiag -P [-v] [-v]
mdiag -q [qosid] [-v]
mdiag -r [reservationid] [-v] [--blocking]
mdiag -R [resourcemanagername] [-v][-v]
mdiag -s [standingreservationid] [--blocking]
mdiag -S [-v] [-v]
mdiag -t [-v] [-v] [partitionid]
mdiag -T [triggerid] [-v][--blocking]
mdiag -u [userid] [-v]
mdiag [--format=xml]
[--about] [--help] [--host=<serverHostName>] [--
loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>]
[--timeout=<seconds>] [--version] [--xml]
```

3.7.4.B Overview

The *mdiag* command is used to display information about various aspects of the cluster and the results of internal diagnostic tests. In summary, it provides the following:

- Current object health and state information
- Current object configuration (resources, policies, attributes, etc.)
- Current and historical performance/utilization information
- Reports on recent failure
- Object messages

Some *mdiag* options gather information from the Moab cache, which prevents them from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler.

3.7.4.C Arguments

Option	Description
-a [accountid]	Display account information.
-b [-l policylevel] [-t partition] [-v]	Display information on jobs blocked by policies, holds, or other factors. If blocked job diagnostics are specified, the `-t` option is also available to constrain the report to analysis of particular partition. Also, with blocked job diagnosis, the `-l` option can be used to specify the analysis policy level.
-c [classid] [-v]	Display class information.
-C [file]	With the vast array of options in the configuration file, the `-C` option does not validate function, but it does analyze the configuration file for syntax errors including use of invalid parameters, deprecated parameters, and some invalid values. If you start Moab with the -e flag, Moab evaluates the configuration file at startup and quits if an error exists. mdiag -C does not print out any #INCLUDE lines listed in moab.cfg (and moab.dat), but it does evaluate and print out the lines found in those included files.
-f [-o user|group|acct|qos|class] [-v] [--flags=relative]	Display fairshare information.
-g [groupid]	Display group information.
-G [Green]	Display green computing information.
-j [jobid] [-t partition] [-v] [-w state|user|account|class|group|qos=VALUE] [--flags=policy] [--blocking]	Display job information.

Option	Description
-l	Diagnose license information contained in the moab.lic file.
-L [-v]	Display limits.
-n [-A creds] [-t partition] [nodeid] [-v]	Display nodes. If node diagnostics are specified, the `-t` option is also available to constrain the report to a particular partition.
-p [-t partition] [-v] [-v]	Display job priority. If priority diagnostics are specified, the `-t` option is also available to constrain the report to a particular partition.
-P [-v] [-v]	Display partition information.
-q [qosid] [-v]	Display qos information.
-r [reservationid] [-v] [--blocking]	Display reservation information.
-R [rmid] [-v] [-v]	Display resource manager information.
-s [srsv] [--blocking]	Display standing reservation information.
-S [-v] [-v]	Display general scheduler information.
-t [-v] [-v] [partitionid]	Display configuration, usage, health, and diagnostic information about partitions maintained by Moab.
-T [triggerid] [-v] [--blocking]	Display trigger information.
-u [userid] [-v]	Display user information.
--format=xml	Display output in XML format.

XML Output

Information for most of the options can be reported as XML also. This is done with the command `mdiag -<option> <CLASS_ID> --format=xml`. For example, XML-based class information will be written to STDOUT in the following format:

```
<Data>
  <class <ATTR>="<VAL>" ... >
    <stats <ATTR>="<VAL>" ... >
      <Profile <ATTR>="<VAL>" ... >
      </Profile>
    </stats>
  </class>
<Data>
  ...
</Data>
```

Of the *mdiag* options, only `-G` and `-L` cannot be reported as XML.

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.1 checkjob
- 3.7.2 checknode

3.7.5 mdiag -a

3.7.5.A Synopsis

mdiag -a [accountid]

3.7.5.B Overview

The *mdiag -a* command provides detailed information about the accounts (a.k.a. projects) Moab is currently tracking. This command also allows an admin to verify correct throttling policies and access provided to and from other credentials.

Example 3-11: Generating information about accounts

```
> mdiag -a
evaluating acct information
Name          Priority        Flags         QDef      QOSList*
PartitionList Target  Limits
engineering        100            -         high      high,urgent,low             [A]
[B]      30.00  MAXJOB=50,75  MAXPROC=400,500
```

```
marketing          1            -          low      low                         [A]
       5.00  MAXJOB=100,110  MAXPS=54000,54500
it                10            -       DEFAULT      DEFAULT,high,urgent,low     [A]
     100.00  MAXPROC=100,1250 MAXPS=12000,12500
  FSWEIGHT=1000
development       100           -         high       high,urgent,low             [A]
[B]      30.00  MAXJOB=50,75 MAXNODE=100,120
research          100           -         high       DEFAULT,high,low            [A]
[B]      30.00  MAXNODE=400,500 MAXPS=900000,1000000
DEFAULT            0            -           -        -                            -
       0.00  -
```

Related Topics

- 2.7.4 Account (or Project) Credential

3.7.6 mdiag -b

3.7.6.A Synopsis

mdiag -b [-l policylevel] [-t partition] [-v]

3.7.6.B Overview

The *mdiag -b* command returns information about blocked jobs.

3.7.7 mdiag -c

3.7.7.A Synopsis

mdiag -c [classid] [-v]

3.7.7.B Overview

The *mdiag -c* command provides detailed information about the classes Moab is currently tracking. This command also allows an admin to verify correct throttling policies and access provided to and from other credentials.

The term class is used interchangeably with the term queue and generally refers to resource manager queue.

3.7.7.C XML Attributes

Name	Description
ADEF	Accounts a class has access to.
CAPACITY	Number of procs available to the class.
DEFAULT.ATTR	Default attributes attached to a job.
DEFAULT.DISK	Default required disk attached to a job.
DEFAULT.FEATURES	Default required node features attached to a job.
DEFAULT.GRES	Default generic resources attached to a job.
DEFAULT.MEM	Default required memory attached to a job.
DEFAULT.NODESET	Default specified nodeset attached to a job.
DEFAULT.WCLIMIT	Default wallclock limit attached to a job.
EXCL.FEATURES	List of excluded (disallowed) node features.
EXCL.FLAGS	List of excluded (disallowed) job flags.
FSTARGET	The class's fairshare target.
HOLD	If TRUE this credential has a hold on it, FALSE otherwise.
HOSTLIST	The list of hosts in this class.
JOBEPILOG	Scheduler level job epilog to be run after job is completed by resource manager (script path).
JOBFLAGS	Default flags attached to jobs in the class.
JOBPROLOG	Scheduler level job prolog to be run before job is started by resource manager (script path).
ID	The unique ID of this class.
LOGLEVEL	The log level attached to jobs in the class.

Name	Description
MAX.PROC	The max processors per job in the class.
MAX.PS	The max processor-seconds per job in the class.
MAX.WCLIMIT	The max wallclock limit per job in the class.
MAXIJOB	The max idle jobs in the class.
MAXIPROC	The max idle processors in the class.
MAXJOBPERUSER	The max jobs per user.
MAXNODEPERJOB	The max nodes per job.
MAXNODEPERUSER	The max nodes per user.
MAXPROCPERJOB	The max processors per job.
MAXPROCPERNODE	The max processors per node.
MAXPROCPERUSER	The max processors per user.
MIN.NODE	The minimum nodes per job in the class.
MIN.PROC	The minimum processors per job in the class.
MIN.WCLIMIT	The minimum wallclock limit per job in the class.
NODEACCESSPOLICY	The node access policy associated with jobs in the class.
OCDPROCFACTOR	Dedicated processor factor.
OCNODE	Overcommit node.
PRIORITY	The class's associated priority.
PRIORITYF	Priority calculation function.
REQ.FEATURES	Required features for a job to be considered in the class.

Name	Description
REQ.FLAGS	Required flags for a job to be considered in the class.
REQ.IMAGE	Required image for a job to be considered in the class.
REQUIREDUSERLIST	The list of users who have access to the class.
RM	The resource manager reporting the class.
STATE	The class's state.
WCOVERRUN	Tolerated amount of time beyond the specified wallclock limit.

Example 3-12: Generating information about classes

Class `fast` has `MAXJOB` soft and hard limits of `100` and `150` respectively and is currently running 8 jobs:

```
> mdiag -c
Class/Queue Status
ClassID        Priority Flags        QDef              QOSList* PartitionList
Target Limits
DEFAULT               0 ---          ---                 ---   ---
0.00  ---
batch                 1 ---          ---                 ---   [A][B]
70.00  MAXJOB=33:200,250
  MAX.WCLIMIT=10:00:00  MAXPROCPERJOB=128
long                  1 ---          low                 low   [A]
10.00  MAXJOB=3:100,200
  MAX.WCLIMIT=1:00:00:00  MAXPROCPERJOB=128
fast                100 ---          high               high   [B]
10.00  MAXJOB=8:100,150
  MAX.WCLIMIT=00:30:00  MAXPROCPERJOB=128
bigmem                1 ---          low,high            low   ---
10.00  MAXJOB=1:100,200
  MAXPROCPERJOB=128
```

The `Limits` column will display limits in the following format: `<USAGE>:<HARDLIMIT>`[,`<SOFTLIMIT>`]

Related Topics

- 3.7.36 showstats - command that displays general statistics

3.7.8 mdiag -f

3.7.8.A Synopsis

mdiag -f [-o user|group|acct|qos|class] [--flags=relative]

3.7.8.B Overview

The *mdiag -f* command is used to display at a glance information about the fairshare configuration and historic resource utilization. The fairshare usage might impact job prioritization, job eligibility, or both based on the credential FSTARGET and FSCAP attributes and by the fairshare priority weights as described in 4.1.1 Job Priority Overview. The information presented by this command includes fairshare configuration and credential fairshare usage over time.

The command hides information about credentials that have no fairshare target and no fairshare cap.

If an object type (<OTYPE>) is specified, then only information for that credential type (user, group, acct, class, or qos) will be displayed. If the relative flag is set, then per user fairshare usage will be displayed relative to each non-user credential (see the second example below).

Relative output is only displayed for credentials that have user mappings. For example, if there is no association between classes and users, no relative per user fairshare usage class breakdown will be provided.

Example 3-13: Standard Fairshare Output

```
> mdiag -f
FairShare Information
Depth: 6 intervals   Interval Length: 00:20:00   Decay Rate: 0.50
FS Policy: DEDICATEDPES
System FS Settings:  Target Usage: 0.00
FSInterval        %     Target       0       1       2       3       4       5
FSWeight       ------- ------- 1.0000  0.5000  0.2500  0.1250  0.0625  0.0312
TotalUsage      100.00 -------    85.3   476.1   478.9   478.5   475.5   482.8
USER
-------------
mattp             2.51 -------    2.20    2.69    2.21    2.65    2.65    3.01
jsmith           12.82 -------   12.66   15.36   10.96    8.74    8.15   13.85
kyliem            3.44 -------    3.93    2.78    4.36    3.11    3.94    4.25
tgh               4.94 -------    4.44    5.12    5.52    3.95    4.66    4.76
walex             1.51 -------    3.14    1.15    1.05    1.61    1.22    1.60
jimf              4.73 -------    4.67    4.31    5.67    4.49    4.93    4.92
poy               4.64 -------    4.43    4.61    4.58    4.76    5.36    4.90
mjackson          0.66 -------    0.35    0.78    0.67    0.77    0.55    0.43
tfw              17.44 -------   16.45   15.59   19.93   19.72   21.38   15.68
```

```
gjohn             2.81 -------    1.66    3.00    3.16    3.06    2.41    3.33
ljill            10.85 -------   18.09    7.23   13.28    9.24   14.76    6.67
kbill            11.10 -------    7.31   14.94    4.70   15.49    5.42   16.61
stevei            1.58 -------    1.41    1.34    2.09    0.75    3.30    2.15
gms               1.54 -------    1.15    1.74    1.63    1.40    1.38    0.90
patw              5.11 -------    5.22    5.11    4.85    5.20    5.28    5.78
wer               6.65 -------    5.04    7.03    7.52    6.80    6.43    2.83
anna              1.97 -------    2.29    1.68    2.27    1.80    2.37    2.17
susieb            5.69 -------    5.58    5.55    5.57    6.48    5.83    6.16
GROUP
-------------
dallas           13.25  15.00   14.61   12.41   13.19   13.29   15.37   15.09
sanjose*          8.86  15.00    6.54    9.55    9.81    8.97    8.35    4.16
seattle          10.05  15.00    9.66   10.23   10.37    9.15    9.94   10.54
austin*          30.26  15.00   29.10   30.95   30.89   28.45   29.53   29.54
boston*           3.44  15.00    3.93    2.78    4.36    3.11    3.94    4.25
orlando*         26.59  15.00   29.83   26.77   22.56   29.49   25.53   28.18
newyork*          7.54  15.00    6.33    7.31    8.83    7.54    7.34    8.24
ACCT
-------------
engineering      31.76  30.00   32.25   32.10   31.94   30.07   30.74   31.14
marketing         8.86   5.00    6.54    9.55    9.81    8.97    8.35    4.16
it                9.12   5.00    7.74    8.65   10.92    8.29   10.64   10.40
development*     24.86  30.00   24.15   24.76   25.00   24.84   26.15   26.78
research         25.40  30.00   29.32   24.94   22.33   27.84   24.11   27.53
QOS
-------------
DEFAULT*          0.00  50.00 ------- ------- ------- ------- ------- -------
high*            83.69  90.00   86.76   83.20   81.71   84.35   83.19   88.02
urgent            0.00   5.00 ------- ------- ------- ------- ------- -------
low*             12.00   5.00    7.34   12.70   14.02   12.51   12.86    7.48
CLASS
-------------
batch*           51.69  70.00   53.87   52.01   50.80   50.38   48.67   52.65
long*            18.75  10.00   16.54   18.36   20.89   18.36   21.53   16.28
fast*            15.29  10.00   18.41   14.98   12.58   16.80   15.15   18.21
bigmem           14.27  10.00   11.17   14.65   15.73   14.46   14.65   12.87
```

An asterisk (*) next to a credential name indicates that that credential has exceeded its fairshare target.

Example 3-14: Grouping User Output by Account

```
> mdiag -f -o acct --flags=relative
FairShare Information
Depth: 6 intervals   Interval Length: 00:20:00   Decay Rate: 0.50
FS Policy: DEDICATEDPES
System FS Settings:  Target Usage: 0.00
FSInterval        %     Target       0       1       2       3       4       5
FSWeight       ------- -------  1.0000  0.5000  0.2500  0.1250  0.0625  0.0312
TotalUsage      100.00 -------    23.8   476.1   478.9   478.5   475.5   482.8
ACCOUNT
-------------
dallas           13.12  15.00   15.42   12.41   13.19   13.29   15.37   15.09
  mattp          19.47 -------   15.00   21.66   16.75   19.93   17.26   19.95
  walex           9.93 -------   20.91    9.28    7.97   12.14    7.91   10.59
  stevei         12.19 -------    9.09   10.78   15.85    5.64   21.46   14.28
  anna           14.77 -------   16.36   13.54   17.18   13.55   15.44   14.37
  susieb         43.64 -------   38.64   44.74   42.25   48.74   37.92   40.81
```

```
sanjose*          9.26  15.00      8.69     9.55     9.81     8.97     8.35     4.16
  mjackson        7.71 -------     6.45     8.14     6.81     8.62     6.54    10.29
  gms            17.61 -------    21.77    18.25    16.57    15.58    16.51    21.74
  wer            74.68 -------    71.77    73.61    76.62    75.80    76.95    67.97
seattle          10.12  15.00     10.16    10.23    10.37     9.15     9.94    10.54
  tgh            49.56 -------    46.21    50.05    53.26    43.14    46.91    45.13
  patw           50.44 -------    53.79    49.95    46.74    56.86    53.09    54.87
austin*          30.23  15.00     25.58    30.95    30.89    28.45    29.53    29.54
  jsmith         42.44 -------    48.77    49.62    35.47    30.70    27.59    46.90
  tfw            57.56 -------    51.23    50.38    64.53    69.30    72.41    53.10
boston*           3.38  15.00      3.78     2.78     4.36     3.11     3.94     4.25
  kyliem        100.00 -------   100.00   100.00   100.00   100.00   100.00   100.00
orlando*         26.20  15.00     30.13    26.77    22.56    29.49    25.53    28.18
  poy            17.90 -------    16.28    17.22    20.30    16.15    20.98    17.39
  ljill          37.85 -------    58.60    26.99    58.87    31.33    57.79    23.67
  kbill          44.25 -------    25.12    55.79    20.83    52.52    21.23    58.94
newyork*          7.69  15.00      6.24     7.31     8.83     7.54     7.34     8.24
  jimf           61.42 -------    69.66    58.94    64.20    59.46    67.21    59.64
  gjohn          38.58 -------    30.34    41.06    35.80    40.54    32.79    40.36
```

Related Topics

- 5.3 Fairshare

3.7.9 mdiag -j

3.7.9.A Synopsis

mdiag -j [jobid] [-t <partition>] [-v] [-w] [--flags=policy] [--xml] [--blocking]

3.7.9.B Overview

The *mdiag -j* command provides detailed information about the state of jobs Moab is currently tracking. This command also performs a large number of sanity and state checks. The job configuration and status information, and the results of the various checks, are presented by this command. The command gathers information from the Moab cache that prevents it from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler. If the `-v` (verbose) flag is specified, additional information about less common job attributes is displayed. If `--flags=policy` is specified, information about job templates is displayed.

If used with the `-t <partition>` option on a running job, the only thing *mdiag -j* shows is if the job is running on the specified partition. If used on a job that is not running, it shows if the job is able to run on the specified partition.

The `-w` flag enables you to specify specific job states, such as Running, Completed, Idle, or ALL (see 2.3.1.A Job States for all valid options), or jobs associated with a given credential (user, acct, class, group, jobgroup, qos). For example:

```
mdiag -j -w user=david            # Displays only David's jobs
mdiag -j -w state=Idle,Running    # Displays only idle or running jobs
mdiag -j -w jobgroup=workflow1    # displays jobs in jobgroup workflow1
```

The *`mdiag -j`* command does not show all subjobs of an array unless you use `mdiag -j --xml`. In the XML, the master job element contains a child element called `ArraySubJobs` that contains the subjobs in the array. Using `mdiag -j -v --xml` shows the completed subjobs also.

3.7.9.C XML Output

If XML output is requested (via the --format=xml argument), XML based node information will be written to STDOUT in the following format:

```
<Data>
  <job ATTR="VALUE" ... > </job>
  ...
</Data>
```

For information about valid attributes, refer to the XML Attributes table.

To show jobs in XML, use `mdiag -j --xml -w [completed=true|system=true|ALL=true]` to limit or filter jobs. This is for XML use only.

Related Topics

- 3.7.1 checkjob
- 3.7.4 mdiag

3.7.10 mdiag -n

3.7.10.A Synopsis

`mdiag -n` `[-t partitionid] [-A creds] [-v] [nodeid]`

3.7.10.B Overview

The *mdiag -n* command provides detailed information about the state of nodes Moab is currently tracking. This command also performs a large number of sanity and state checks. The node configuration and status information and the results of the various checks are presented by this command.

3.7.10.C Arguments

Flag	Argument	Description
[-A]	{user|group|account|qos|class|job}: `<OBJECTID>`	Report if each node is accessible by requested job or credential.
	`[nodeid]`	Report on the specified node (default is all nodes).
[-t]	`<partitionid>`	Report only nodes from specified partition.
[-v]	---	Show verbose output (do not truncate columns and add columns for additional node attributes).

3.7.10.D Output

This command presents detailed node information in whitespace-delineated fields.

The output of this command can be extensive and the values for a number of fields can be truncated. If truncated, the `-v` flag can be used to display full field content.

Column	Format
Name	`<NODE NAME>`
State	`<NODE STATE>`
Procs	`<AVAILABLE PROCS>:<CONFIGURED PROCS>`
Memory	`<AVAILABLE MEMORY>:<CONFIGURED MEMORY>`
Disk	`<AVAILABLE DISK>:<CONFIGURED DISK>`
Swap	`<AVAILABLE SWAP>:<CONFIGURED SWAP>`

Column	Format
Speed	`<RELATIVE MACHINE SPEED>`
Opsys	`<NODE OPERATING SYSTEM>`
Arch	`<NODE HARDWARE ARCHITECTURE>`
Par	`<PARTITION NODE IS ASSIGNED TO>`
Load	`<CURRENT 1 MINUTE BSD LOAD>`
Rsv	`<NUMBER OF RESERVATIONS ON NODE>`
Classes	`<CLASS NAME>`
Network	`<NETWORK NAME>`...
Features	`<NODE FEATURE>`...

3.7.10.E Examples

```
> mdiag -n

compute node summary
Name                    State   Procs      Memory         Opsys

opt-001                  Busy    0:2      2048:2048        SUSE
opt-002                  Busy    0:2      2048:2048        SUSE
opt-003                  Busy    0:2      2048:2048        SUSE
opt-004                  Busy    0:2      2048:2048        SUSE
opt-005                  Busy    0:2      2048:2048        SUSE
opt-006                  Busy    0:2      2048:2048        SUSE
WARNING:   swap is low on node opt-006
opt-007                  Busy    0:2      2048:2048        SUSE
opt-008                  Busy    0:2      2048:2048        SUSE
opt-009                  Busy    0:2      2048:2048        SUSE
opt-010                  Busy    0:2      2048:2048        SUSE
opt-011                  Busy    0:2      2048:2048        SUSE
opt-012                  Busy    0:2      2048:2048        SUSE
opt-013                  Busy    0:2      2048:2048        SUSE
opt-014                  Busy    0:2      2048:2048        SUSE
opt-015                  Busy    0:2      2048:2048        SUSE
opt-016                  Busy    0:2      2048:2048        SUSE
x86-001                  Busy    0:1       512:512       Redhat
x86-002                  Busy    0:1       512:512       Redhat
x86-003                  Busy    0:1       512:512       Redhat
x86-004                  Busy    0:1       512:512       Redhat
x86-005                  Idle    1:1       512:512       Redhat
x86-006                  Idle    1:1       512:512       Redhat
x86-007                  Idle    1:1       512:512       Redhat
```

```
x86-008                   Busy    0:1       512:512       Redhat
x86-009                   Down    1:1       512:512       Redhat
x86-010                   Busy    0:1       512:512       Redhat
x86-011                   Busy    0:1       512:512       Redhat
x86-012                   Busy    0:1       512:512       Redhat
x86-013                   Busy    0:1       512:512       Redhat
x86-014                   Busy    0:1       512:512       Redhat
x86-015                   Busy    0:1       512:512       Redhat
x86-016                   Busy    0:1       512:512       Redhat
P690-001                  Busy    0:1     16384:16384        AIX
P690-002                  Busy    0:1     16384:16384        AIX
P690-003                  Busy    0:1     16384:16384        AIX
P690-004                  Busy    0:1     16384:16384        AIX
P690-005                  Busy    0:1     16384:16384        AIX
P690-006                  Busy    0:1     16384:16384        AIX
P690-007                  Idle    1:1     16384:16384        AIX
P690-008                  Idle    1:1     16384:16384        AIX
WARNING:   node P690-008 is missing ethernet adapter
P690-009                  Busy    0:1     16384:16384        AIX
P690-010                  Busy    0:1     16384:16384        AIX
P690-011                  Busy    0:1     16384:16384        AIX
P690-012                  Busy    0:1     16384:16384        AIX
P690-013                  Busy    0:1     16384:16384        AIX
P690-014                  Busy    0:1     16384:16384        AIX
P690-015                  Busy    0:1     16384:16384        AIX
P690-016                  Busy    0:1     16384:16384        AIX
-----                      ---    6:64   745472:745472     -----

Total Nodes: 36  (Active: 30  Idle: 5  Down: 1)
```

Warning messages are interspersed with the node configuration information with all warnings preceded by the keyword WARNING.

3.7.10.F XML Output

If XML output is requested (via the --format=xml argument), XML based node information will be written to STDOUT in the following format:

```
mdiag -n --format=xml
<Data>
  <node> <ATTR>="<VAL>" ... </node>
  ...
</Data>
```

3.7.10.G XML Attributes

Name	Description
ACL	Node Access Control List.

Name	Description
AGRES	Available generic resources.
ALLOCRES	Special allocated resources (like VLANs).
ARCH	The node's processor architecture.
AVLCLASS	Classes available on the node.
AVLETIME	Time when the node will no longer be available (used in Utility centers).
AVLSTIME	Time when the node will be available (used in Utility centers).
CFGCLASS	Classes configured on the node.
ENABLEPROFILING	If true, a node's state and usage is tracked over time.
FEATURES	A list of comma-separated custom features describing a node.
GEVENT	A user-defined event that allows Moab to perform some action.
GMETRIC	A list of comma-separated consumable resources associated with a node.
GRES	generic resources on the node.
HOPCOUNT	How many hops the node took to reach this Moab (used in hierarchical grids).
ISDELETED	Node has been deleted.
ISDYNAMIC	Node is dynamic (used in Utility centers).
JOBLIST	The list of jobs currently running on a node.
LOAD	Current load as reported by the resource manager.
LOADWEIGHT	Load weight used when calculating node priority.
MAXJOB	See 10.4 Node Specific Policies for details.

Name	Description
MAXJOBPERUSER	See 10.4 Node Specific Policies for details.
MAXLOAD	See 10.4 Node Specific Policies for details.
MAXPROC	See 10.4 Node Specific Policies for details.
MAXPROCPERUSER	See 10.4 Node Specific Policies for details.
NETWORK	The ability to specify which networks are available to a given node is limited to only a few resource managers. Using the `NETWORK` attribute, admins can establish this node to network connection directly through the scheduler. The NODECFG parameter allows this list to be specified in a comma-delimited list.
NODEID	The unique identifier for a node.
NODESTATE	The state of a node.
OS	A node's operating system.
OSLIST	Operating systems the node can run.
OSMODACTION	URL for changing the operating system.
OWNER	Credential type and name of owner.
PARTITION	The partition a node belongs to. See 10.2 Node Location for details.
POWER	The state of the node's power. Either ON or OFF.
PRIORITY	The fixed node priority relative to other nodes.
PROCSPEED	A node's processor speed information specified in MHz.
RACK	The rack associated with a node's physical location.
RADISK	The total available disk on a node.
RAMEM	The total available memory on a node.

Name	Description
RAPROC	The total number of processors available on a node.
RASWAP	The total available swap on a node.
RCMEM	The total configured memory on a node.
RCPROC	The total configured processors on a node.
RCSWAP	The total configured swap on a node.
RequestID	Dynamic Node RequestID set by the resource manager.
RESCOUNT	Number of reservations on the node.
RESOURCES	Deprecated (use GRES).
RSVLIST	List of reservations on the node.
RMACCESSLIST	A comma-separated list of resource managers who have access to a node.
SIZE	The number of slots or size units consumed by the node.
SLOT	The first slot in the rack associated with the node's physical location.
SPEED	A node's relative speed.
SPEEDWEIGHT	Speed weight used to calculate node's priority.
STATACTIVETIME	Time node was active.
STATMODIFYTIME	Time node's state was modified.
STATTOTALTIME	Time node has been monitored.
STATUPTIME	Time node has been up.
TASKCOUNT	The number of tasks on a node.
TTL	Dynamic Node Time To Live set by the resource manager (expiration date in epoch format).

Related Topics

- 3.7.2 checknode

3.7.11 mdiag -p

3.7.11.A Synopsis

```
mdiag -p [-t partition] [-v] [-v]
```

3.7.11.B Overview

The `mdiag -p` command is used to display at a glance information about the job priority configuration and its effects on the current eligible jobs. The information presented by this command includes priority weights, priority components, and the percentage contribution of each component to the total job priority.

The command hides information about priority components that have been deactivated (i.e., by setting the corresponding component priority weight to 0). For each displayed priority component, this command gives a small amount of context sensitive information. The following table documents this information. In all cases, the output is of the form `<PERCENT>(<CONTEXT INFO>)`, where `<PERCENT>` is the percentage contribution of the associated priority component to the job's total priority.

By default, this command only shows information for jobs that are eligible for immediate execution. Jobs that violate soft or hard policies, or have holds, job dependencies, or other job constraints in place will not be displayed. If priority information is needed for any of these jobs, use the -v flag or the checkjob command.

3.7.11.C Format

Flag	Name	Format	Default	Description	Example
-t		-t partition	all partitions	Constrain the report to a particular partition.	`> mdiag -p -t partition1` *Display priority summary information for jobs in*

Flag	Name	Format	Default	Description	Example
					partition1.
-v	VERBOSE	---	---	Display verbose priority information. If specified, display priority breakdown information for blocked, eligible, and active jobs. By default, only information for eligible jobs is displayed. To view blocked jobs in addition to eligible, run *`mdiag -p -v -v`*.	`> mdiag -p -v` *Display priority summary information for eligible and active jobs.*

3.7.11.D Output

Priority Component	Format	Description
Target	`<PERCENT>`()	
QOS	`<PERCENT>` (`<QOS>`:`<QOSPRI>`)	`QOS` — QOS associated with job `QOSPRI` — Priority assigned to the QOS
FairShare	`<PERCENT>` (`<USR>` : `<GRP>` :`<ACC>`:`<QOS>`:`<CLS>`)	`USR` — user fs usage - user fs target `GRP` — group fs usage - group fs target `ACC` — account fs usage - account fs target `QOS` — QOS fs usage - QOS fs target `CLS` — class fs usage - class fs target
Service	`<PERCENT>` (`<QT>`:`<XF>`:`<Byp>`)	`QTime` — job queue time that is applicable towards priority (in minutes) `XF` — current theoretical minimum XFactor is job were to start immediately `Byp` — number of times job was bypassed

Priority Component	Format	Description
		by lower priority jobs via backfill
Resource	`<PERCENT>` (`<NDE>:<PE>:<PRC>:<MEM>`)	`NDE` — nodes requested by job `PE` — Processor Equivalents as calculated by all resources requested by job `PRC` — processors requested by job `MEM` — real memory requested by job

3.7.11.E Examples

Example 3-15: mdiag -p

```
diagnosing job priority information (partition: ALL)

Job                     PRIORITY*   Cred(  QOS)     FS(Accnt)   Serv(QTime)
Weights   --------       1(     1)      1(     1)     1(     1)

13678                       1321*     7.6(100.0)    0.2(   2.7)   92.2(1218.)
13698                        235*    42.6(100.0)    1.1(   2.7)   56.3(132.3)
13019                       8699      0.6( 50.0)    0.3( 25.4)    99.1(8674.)
13030                       8699      0.6( 50.0)    0.3( 25.4)    99.1(8674.)
13099                       8537      0.6( 50.0)    0.3( 25.4)    99.1(8512.)
13141                       8438      0.6( 50.0)    0.2( 17.6)    99.2(8370.)
13146                       8428      0.6( 50.0)    0.2( 17.6)    99.2(8360.)
13153                       8360      0.0(   1.0)   0.1( 11.6)    99.8(8347.)
13177                       8216      0.0(   1.0)   0.1( 11.6)    99.8(8203.)
13203                       8127      0.6( 50.0)    0.3( 25.4)    99.1(8102.)
13211                       8098      0.0(   1.0)   0.1( 11.6)    99.8(8085.)
...
13703                        137     36.6( 50.0)   12.8( 17.6)    50.6( 69.2)
13702                         79      1.3(   1.0)   5.7(   4.5)   93.0( 73.4)

Percent Contribution   --------       0.9(   0.9)   0.4(   0.4)   98.7( 98.7)

* indicates system prio set on job
```

The *`mdiag -p`* command only displays information for priority components actually utilized. In the above example, QOS, Account Fairshare, and QueueTime components are utilized in determining a job's priority. Other components, such as Service Targets, and Bypass are not used and therefore are not displayed. See 4.1.1 Job Priority Overview for more information. The output consists of a header, a job by job analysis of jobs, and a summary section.

The header provides column labeling and provides configured priority component and subcomponent weights. In the above example, `QOSWEIGHT` is set to `1000` and `FSWEIGHT` is set to `100`. When configuring fairshare, you also have the option of weighting the individual components of a job's overall fairshare, including its user, group, and account fairshare components. In this output, the QoS and account fairshare weights are set to `1`.

The job by job analysis displays a job's total priority and the percentage contribution to that priority of each of the priority components. In this example, job `13019` has a total priority of 8699. Both QOS and Fairshare contribute to the job's total priority although these factors are quite small, contributing `0.6%` and `0.3%` respectively with the fairshare factor being contributed by an account fairshare target. For this job, the dominant factor is the `service` subcomponent `qtime`, which is contributing `99.1%` of the total priority since the job has been in the queue for approximately 8600 minutes.

At the end of the job by job description, a Totals line is displayed, which documents the average percentage contributions of each priority component to the current idle jobs. In this example, the QOS, Fairshare, and Service components contributed an average of `0.9%`, `0.4%`, and `98.7%` to the jobs' total priorities.

Related Topics

- 4.1.1 Job Priority Overview

3.7.12 mdiag -q

3.7.12.A Synopsis

`mdiag -q` `[qosid]`

3.7.12.B Overview

The *`mdiag -q`* command is used to present information about each QOS maintained by Moab. The information presented includes QOS name, membership, scheduling priority, weights and flags.

3.7.12.C Examples

Example 3-16: Standard QOS Diagnostics

```
> mdiag -q
QOS Status
System QOS Settings:  QList: DEFAULT (Def: DEFAULT)  Flags: 0
Name                  * Priority QTWeight QTTarget XFWeight XFTarget     QFlags
JobFlags Limits
DEFAULT                        1        1        3        1     5.00   PREEMPTEE
[NONE] [NONE]
```

```
  Accounts:  it research
  Classes:  batch
[ALL]                         0         0         0         0      0.00      [NONE]
[NONE] [NONE]
high                       1000         1         2         1     10.00   PREEMPTOR
[NONE] [NONE]
  Accounts:  engineering it development research
  Classes:  fast
urgent                    10000         1         1         1      7.00   PREEMPTOR
[NONE] [NONE]
  Accounts:  engineering it development
low                         100         1         5         1      1.00   PREEMPTEE
[NONE] [NONE]
  Accounts:  engineering marketing it development research
  Classes:  long bigmem
```

3.7.13 mdiag -r

3.7.13.A Synopsis

`mdiag -r [reservationid] [-v] [--blocking]`

3.7.13.B Overview

The `mdiag -r` command enables admins to look at detailed reservation information. It provides the name, type, partition, starttime and endtime, proc and node counts, and actual utilization figures. It also provides detailed information about which resources are being used, how many nodes, how much memory, swap, and processors are being associated with each task. Admins can also view the Access Control Lists for each reservation, and any flags that may be active in the reservation. The command gathers information from the Moab cache, which prevents it from waiting for the scheduler, but the `--blocking` option can be used to bypass the cache and allow waiting for the scheduler.

3.7.13.C Examples

```
> mdiag -r
Diagnosing Reservations
RsvID                      Type  Par   StartTime     EndTime     Duration Node Task
Proc
-----                      ---- ---   ---------     -------     -------- ---- ---- ---
-
engineer.0.1               User   A    -6:29:00    INFINITY     INFINITY    0    0
7
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==batch+:==long+:==fast+:==bigmem+ QOS==low-:==high+ JATTR==PREEMPTEE+
    CL:    RSV==engineer.0.1
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr10n01 fr10n03 fr10n05 fr10n07 fr10n09 fr10n11 fr10n13
fr10n15')
```

```
    PH Allocated to Jobs: 43.77/45.44 (96.31%)
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 1:00:00:00  Days: ALL)
research.0.2                User   A    -6:29:00    INFINITY     INFINITY    0    0
8
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==batch+:==long+:==fast+:==bigmem+ QOS==high+:==low- JATTR==PREEMPTEE+
    CL:    RSV==research.0.2
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr3n01 fr3n03 fr3n05 fr3n07 fr3n07 fr3n09 fr3n11 fr3n13
fr3n15')
    PH Allocated to Jobs: 51.60/51.93 (99.36%)
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 1:00:00:00  Days: ALL)
fast.0.3                    User   A    00:14:05     5:14:05      5:00:00    0    0
16
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==fast+ QOS==high+:==low+:==urgent+:==DEFAULT+ JATTR==PREEMPTEE+
    CL:    RSV==fast.0.3
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr12n01 fr12n02 fr12n03 fr12n04 fr12n05 fr12n06 fr12n07
fr12n08 fr12n09 fr12n10 fr12n11 fr12n12 fr12n13 fr12n14 fr12n15 fr12n16')
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 5:00:00  Days:
Mon,Tue,Wed,Thu,Fri)
fast.1.4                    User   A  1:00:14:05  1:05:14:05      5:00:00    0    0
16
    Flags: STANDINGRSV IGNSTATE OWNERPREEMPT
    ACL:   CLASS==fast+ QOS==high+:==low+:==urgent+:==DEFAULT+ JATTR==PREEMPTEE+
    CL:    RSV==fast.1.4
    Task Resources: PROCS: [ALL]
    Attributes (HostExp='fr12n01 fr12n02 fr12n03 fr12n04 fr12n05 fr12n06 fr12n07
fr12n08 fr12n09 fr12n10 fr12n11 fr12n12 fr12n13 fr12n14 fr12n15 fr12n16')
    SRAttributes (TaskCount: 0  StartTime: 00:00:00  EndTime: 5:00:00  Days:
Mon,Tue,Wed,Thu,Fri)
job2411                      Job   A   -00:01:00    00:06:30     Each tile contains a
summary information about the service it represents, including the following:
    ACL:   JOB==job2411=
    CL:    JOB==job2411 USER==jimf GROUP==newyork ACCT==it CLASS==bigmem QOS==low
JATTR==PREEMPTEE DURATION==00:07:30 PROC==6 PS==2700
job1292                      Job   A    00:00:00    00:07:30     00:07:30    0    0
4
    ACL:   JOB==job1292=
    CL:    JOB==job1292 USER==jimf GROUP==newyork ACCT==it CLASS==batch QOS==DEFAULT
JATTR==PREEMPTEE DURATION==00:07:30 PROC==4 PS==1800
```

With the `-v` option, a nodes line is included for each reservation and shows how many nodes are in the reservation, and how many tasks are on each node:

```
> mdiag -r -v
Diagnosing Reservations
RsvID                       Type Par   StartTime     EndTime     Duration Node Task
Proc
-----                       ---- ---   ---------     -------     -------- ---- ---- ---
-
Moab.6                       Job   B   -00:01:05    00:00:35     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.6=
    CL:    JOB==Moab.6 USER==tuser1 GROUP==tgroup1 CLASS==fast QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node002:1'
    Rsv-Group: Moab.6
```

```
Moab.4                     Job   B   -00:01:05    00:00:35     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.4=
    CL:    JOB==Moab.4 USER==tuser1 GROUP==tgroup1 CLASS==batch QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node002:1'
    Rsv-Group: Moab.4

Moab.5                     Job   A   -00:01:05    00:00:35     00:01:40    3    3
6
    Flags: ISACTIVE
    ACL:   JOB==Moab.5=
    CL:    JOB==Moab.5 USER==tuser1 GROUP==tgroup1 ACCT==marketing CLASS==long
QOS==low JPRIORITY<=0 DURATION==00:01:40 PROC==6 PS==600
    Task Resources: PROCS: [ALL]
    SubType: JobReservation
    Nodes='node008:1,node007:1,node006:1'
    Rsv-Group: Moab.5

Moab.7                     Job   A   -00:01:04    00:00:36     00:01:40    1    1
1
    Flags: ISACTIVE
    ACL:   JOB==Moab.7=
    CL:    JOB==Moab.7 USER==tuser1 GROUP==tgroup1 CLASS==bigmen QOS==starter
JPRIORITY<=0 DURATION==00:01:40 PROC==1 PS==100
    SubType: JobReservation
    Nodes='node005:1'
    Rsv-Group: Moab.7

Moab.2                     Job   A   -00:01:07     3:58:53      4:00:00    1    2
2
    Flags: ISACTIVE
    ACL:   JOB==Moab.2=
    CL:    JOB==Moab.2 USER==tuser1 GROUP==tgroup1 QOS==starter JPRIORITY<=0
DURATION==4:00:00 PROC==2 PS==28800
    SubType: JobReservation
    Nodes='node009:1'
    Rsv-Group: Moab.2

Moab.8                     Job   A     3:58:53     7:58:53      4:00:00    8   16
16
    Flags: PREEMPTEE
    ACL:   JOB==Moab.8=
    CL:    JOB==Moab.8 USER==tuser1 GROUP==tgroup1 ACCT==development CLASS==bigmen
QOS==starter JPRIORITY<=0 DURATION==4:00:00 PROC==16 PS==230400
    SubType: JobReservation

Nodes='node009:1,node008:1,node007:1,node006:1,node005:1,node004:1,node003:1,node001:
1'
    Attributes (Priority=148)
    Rsv-Group: idle

system.3                  User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.3=
    CL:    RSV==system.3
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node254:1'
    Attributes (HostExp='node254')
    PH Allocated to Jobs: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2


system.2                   User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.2=
    CL:    RSV==system.2
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node255:1'
    Attributes (HostExp='node255')
    PH Allocated to Jobs: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2


system.1                   User bas   -00:01:08    INFINITY     INFINITY    1    1
2
    Flags: ISCLOSED,ISACTIVE
    ACL:   RSV==system.1=
    CL:    RSV==system.1
    Accounting Creds:  User:root
    Task Resources: PROCS: [ALL]
    SubType: Other
    Nodes='node256:1'
    Attributes (HostExp='node256')
    PH Allocated to Jobs: 0.00/0.01 (0.00%)
    History: 1322773208:PROCS=2
```

3.7.14 mdiag -R

3.7.14.A Synopsis

mdiag -R [-v] [-v] [resourcemanagerid]

3.7.14.B Overview

The *mdiag -R* command is used to present information about configured resource managers. The information presented includes name, host, port, state, type, performance statistics and failure notifications.

3.7.14.C Examples

```
> $ mdiag -R -v
diagnosing resource managers

RM[internal]  State: ---  Type: SSS  ResourceType: COMPUTE
  Max Fail/Iteration: 0
  JobCounter:         6
  Partition:          SHARED
  RM Performance:     AvgTime=0.00s  MaxTime=0.00s  (55353 samples)
  RM Languages:       -
  RM Sub-Languages:   -

RM[torque]    State: Active  Type: PBS  ResourceType: COMPUTE
  Timeout:            30000.00 ms
  Version:            '7.1.0'
  Job Submit URL:     exec:///opt/torque-7.1.0/bin/qsub
  Objects Reported:   Nodes=1 (12 procs)  Jobs=1
  Nodes Reported:     1 (N/A)
  Flags:              executionServer
  Partition:          torque
  Event Management:   EPORT=15004  (last event: 00:03:07)
  NOTE:  SSS protocol enabled
  Submit Command:     /opt/torque-7.1.0/bin/qsub
  DefaultClass:       batch
  Total Jobs Started: 1
  RM Performance:     AvgTime=0.00s  MaxTime=35.00s  (220097 samples)
  RM Languages:       PBS
  RM Sub-Languages:   PBS

RM[torque] Failures:
  clusterquery     (683 of 55349 failed)
        -12days  'cannot connect to PBS server '' (pbs_errno=15033, 'Batch protocol
error')'

NOTE:  use 'mrmctl -f messages <RMID>' to clear stats/failures

RM[FLEXlm]    State: Active  Type: NATIVE  ResourceType: LICENSE
  Timeout:            30000.00 ms
  Cluster Query URL:  exec://$TOOLSDIR/flexlm/license.mon.flexLM.pl
  Licenses Reported:  6 types (250 of 282 available)
  Partition:          SHARED
  License Stats:      Avg License Avail:   239.01  (978 iterations)
  Iteration Summary:  Idle: 396.42  Active: 150.92  Busy: -447.34
  License biocol          50 of  50 available  (Idle: 100.00%  Active: 0.00%)
  License cloudform       100 of 100 available  (Idle: 100.00%  Active: 0.00%)
  License mathworks       8 of  25 available  (Idle: 52.00%  Active: 48.00%)
  License verity          25 of  25 available  (Idle: 100.00%  Active: 0.00%)
  Event Management:   (event interface disabled)
  RM Performance:     AvgTime=0.00s  MaxTime=0.61s  (1307618 samples)
        clusterquery:   AvgTime=0.02s  MaxTime=0.61s  (9465 samples)
          queuequery:   AvgTime=0.00s  MaxTime=0.00s  (1 samples)
        rminitialize:   AvgTime=0.00s  MaxTime=0.00s  (1 samples)
             getdata:   AvgTime=0.17s  MaxTime=0.60s  (978 samples)
  RM Languages:       NATIVE
  RM Sub-Languages:   NATIVE

AM[mam]  Type: MAM  State: 'Active'
  Host:                      localhost
  Port:                      7112
  Timeout:                   15
  Accounting Mode:           strict-allocation
  Job Charge Policy:         All
  Reservation Charge Policy: Select
  Retry Failed Charges:      TRUE

AM[mam] Failures:
  Thu Jun 21 14:32:45  Create           'Failure registering job Create (1) with
accounting manager -- server rejected request with status code 740 - Insufficient
funds: There are no valid allocations to satisfy the quote'
```

3.7.15 mdiag -s

3.7.15.A Synopsis

mdiag -s [reservationid] [-v]>]

3.7.15.B Overview

The *mdiag -s* command enables admins to look at detailed standing reservation information. It provides the name, type, partition, starttime and endtime, period, task count, host list, and a list of child instances.

3.7.15.C Examples

```
> mdiag -s
standing reservation overview
RsvID                    Type     Par      StartTime     EndTime     Duration     Period
-----                    ----     ---      ---------     -------     --------     ------

TestSR                   User     ---      00:00:00          ---     00:00:00        DAY
  Days:                  ALL
  Depth:         2
  RsvList:       testSR.1,testSR.2,testSR.3
  HostExp:       'node1,node2,node4,node8'

test2                    User     ---      00:00:00          ---     00:00:00        DAY
  Days:          ALL
  TaskCount:     4
  Depth:         1
  RsvList:       test2.4,test2.5
```

3.7.16 mdiag -S

3.7.16.A Synopsis

mdiag -S [-v] [-v]

3.7.16.B Overview

The *mdiag -S* command is used to present information about the status of the scheduler and grid interface.

This command will report on the following aspects of scheduling:

- General Scheduler Configuration
  - Reports short and long term scheduler load
  - Reports detected overflows of node, job, reservation, partition, and other scheduler object tables
- High Availability
  - Configuration
  - Reports health of HA primary
  - Reports health of HA backup
- Scheduling Status
  - Reports if scheduling is paused
  - Reports if scheduling is stopped
- System Reservation Status
  - Reports if global system reservation is active
- Message Profiling/Statistics Status
- Moab scheduling activities (only with *`mdiag -S -v -v`*)
  - Activity[JobStart]: Time Moab spends telling the resource manager to start a job and waiting for a response
  - Activity[RMResourceLoad]: Time Moab spends querying license managers and nodes
  - Activity[RMWorkloadLoad]: Time Moab spends querying resource managers about jobs (as opposed to nodes)
  - Activity[Schedule]: Time Moab spends prioritizing jobs and scheduling them onto nodes
  - Activity[UIProcess]: Time Moab spends handling client commands

3.7.16.C Example

```
> mdiag -S
Moab Server running on orion-1:43225  (Mode: NORMAL)
  Load(5m)  Sched: 12.27%  RMAction: 1.16%  RMQuery: 75.30%  User: 0.29%  Idle: 10.98%
  Load(24h) Sched: 10.14%  RMAction: 0.93%  RMQuery: 74.02%  User: 0.11%  Idle: 13.80%
  HA Fallback Server:  orion-2:43225  (Fallback is Ready)
  Note:  system reservation blocking all nodes
  Message:  profiling enabled (531 of 600 samples/5:00 interval)
```

3.7.17 mdiag -t

3.7.17.A Synopsis

`mdiag -t` [-v] [-v] [`partitionid`]

3.7.17.B Overview

The `mdiag -t` command is used to present configuration, usage, health, and diagnostic information about partitions maintained by Moab. The information presented includes partition name, limits, configured and available resources, allocation weights and policies.

3.7.17.C Example

Example 3-17: Standard partition diagnostics

```
> mdiag -t
Partition Status
...
```

3.7.18 mdiag -T

3.7.18.A Synopsis

`mdiag -T` [`triggerid`] [-v] [--blocking]

3.7.18.B Overview

The `mdiag -T` command is used to present information about each Trigger. The information presented includes TrigID, Object ID, Event (Etype) TType, Attype, ActionDate, State. The command gathers information from the Moab cache, which prevents it from waiting for the scheduler, but the `--blocking` option can be used to bypass the cache and allow waiting for the scheduler.

3.7.18.C Examples

```
> mdiag -T
TrigID                  Object ID                 Event  TType    AType ActionDate
 State
--------------------- -------------------- -------- -------- ----- -------------- ----
-------
```

```
sched_trig.0          sched:Moab                end generic   exec              -
Blocked
3                     node:node010         threshol generic   exec              -
Blocked
5                     job:Moab.7            preempt generic   exec              -
Blocked
6                     job:Moab.8            preempt generic   exec              -
Blocked
7                     qos:HIGH             threshol elastic   exec              -
Blocked
4*                    job:Moab.5              start generic   exec        0:00:36
Failure
* indicates trigger has completed
```

Example 3-18:

```
> mdiag -T -v
TrigID                Object ID                Event TType     AType
ActionDate       State
--------------------- -------------------- --------- --------- -------  -----------------
---- -----------
sched_trig.0          sched:Moab                end generic    exec
-     Blocked
  Name:           sched_trig
  Flags:          globaltrig
  BlockUntil:     INFINITY  ActiveTime:  ---
  Action Data:    date
  NOTE:  trigger can launch

3                     node:node010         threshol generic    exec
-     Blocked
  Flags:          globaltrig
  BlockUntil:     INFINITY  ActiveTime:  ---
  Threshold:      CPULoad > 3.00  (current value: 0.00)
  Action Data:    date
  NOTE:  trigger cannot launch - threshold not satisfied - threshold type not
supported

5                     job:Moab.7            preempt generic    exec
-     Blocked
  Flags:          user,globaltrig
  BlockUntil:     INFINITY  ActiveTime:  ---
  Action Data:    $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME

6                     job:Moab.8            preempt generic    exec
-     Blocked
  Flags:          user,globaltrig
  BlockUntil:     INFINITY  ActiveTime:  ---
  Action Data:    $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME
  NOTE:  trigger cannot launch - parent job Moab.8 is in state Idle

7                     qos:HIGH             threshol elastic    exec
-     Blocked
  Flags:          multifire,globaltrig
  BlockUntil:     INFINITY  ActiveTime:  ---
  Timeout:        00:05:00
  Threshold:      BacklogCompletionTime > 500.00  (current value: 0.00)
  Trigger Type:   elastic
  RearmTime:      00:00:10
  Action Data:    $HOME/geometry.pl $REQUESTGEOMETRY
  NOTE:  trigger cannot launch - threshold not satisfied - threshold not satisfied -
requires usage 0.000000 > 500.000000
```

```
4*                    job:Moab.5              start generic    exec  Mon Jan 16
12:33:00     Failure
  Launch Time:    -00:02:17
  Flags:          globaltrig
  Last Execution State: Failure (ExitCode: 0)
  BlockUntil:     00:00:00  ActiveTime:  00:00:00
  Action Data:    $HOME/tools/preemptnotify.pl $OID $OWNER $HOSTNAME
  ALERT:  trigger failure detected
  Message:        'exec '/usr/test/moab/tools/preemptnotify.pl' cannot be located or is
not executable'

* indicates trigger has completed
```

3.7.19 mdiag -u

3.7.19.A Synopsis

mdiag -u [userid]

3.7.19.B Overview

The *mdiag -u* command is used to present information about user records maintained by Moab. The information presented includes user name, UID, scheduling priority, default job flags, default QOS level, List of accessible QOS levels, and list of accessible partitions.

3.7.19.C Examples

```
> mdiag -u
evaluating user information
Name             Priority        Flags         QDef      QOSList*        PartitionList
Target  Limits

jvella                0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Engineering
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
[NONE]                0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
reynolds              0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Administration
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
mshaw                 0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Test
  Message:  profiling enabled (584 of 3000 samples/00:15:00 interval)
kforbes               0       [NONE]       [NONE]       [NONE]                [NONE]
0.00  [NONE]
  ALIST=Shared
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
```

```
gastor               0        [NONE]        [NONE]        [NONE]                 [NONE]
0.00   [NONE]
  ALIST=Engineering
  Message:  profiling enabled (597 of 3000 samples/00:15:00 interval)
```

Note that only users who have jobs that are currently queued or have been queued since Moab was most recently started are listed.

Related Topics

- 3.7.36 showstats - command that displays user statistics

3.7.20 mjobctl

3.7.20.A Synopsis

```
mjobctl -c jobexp
mjobctl -c -w [jobexp] attr=val
mjobctl -C jobexp
mjobctl -e jobid
mjobctl -F jobexp
mjobctl -h [User|System|Batch|Defer|All] jobexp
mjobctl -m attr{+=|=|-=}val jobexp [--flags=force]
mjobctl -N [<SIGNO>] jobexp
mjobctl -p <PRIORITY> jobexp
mjobctl -q {diag|starttime|hostlist} jobexp
mjobctl -r jobexp
mjobctl -R jobexp [--flags=force | unmigrate
mjobctl -s jobexp
mjobctl -u jobexp
mjobctl -w attr{+=|=|-=}val jobexp
mjobctl -x [-w flags=val jobexp
[--about] [--help] [--host=<serverHostName>] [--
loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>]
[--timeout=<seconds>] [--version] [--xml]
```

3.7.20.B Overview

The *mjobctl* command controls various aspects of jobs. It is used to submit, cancel, execute, and checkpoint jobs. It can also display diagnostic information about each job. The *mjobctl* command enables the Moab Admin to control almost all aspects of job behavior. See Chapter 9: General Job Administration for more information on jobs and their attributes.

3.7.20.C Options

-c - Cancel	
Format	JOBEXP
Description	Cancel a job. Use -w (following a -c flag) to specify job cancellation according to given credentials or job attributes. See -c -w for more information. You can use `mjobctl -c flags=follow-dependency <job_id>` to cancel all jobs that the `<job_id>` depends on. If you want to cancel all jobs that depend on this `<job_id>`, add `FLAGS=CANCELFAILEDDEPENDENCYJOBS` to your SCHEDCFG entry in moab.cfg file. See the flag CANCELFAILEDDEPENDENCYJOBS for more information.
Example	`> mjobctl -c job1045` *Cancel job `job1045`.*

-c -w - Cancel Where	
Format	`[JOBEXP] <ATTR>=<VALUE>` Where `<ATTR>`=[user | account | qos | class | reqreservation (RsvName) | state (JobState) | jobname (JobName, not job ID)] | partition
Description	Cancel a job based on a given credential or job attribute. Use `-w` following a -c flag to specify job cancellation according to credentials or job attributes (see examples). See 2.3.1.A Job States for a list of all job states. Also, you can cancel jobs from given partitions using `-w`

-c -w - Cancel Where	
	`partition=<PAR1>[<PAR2>...]]`; however, you must also either use another `-w` flag to specify a job or use the standard job expression.

Example	`> mjobctl -c -w state=USERHOLD`
	Cancels all jobs that currently have a `USERHOLD` *on them.*
	`> mjobctl -c -w user=user1 -w acct=acct1`
	Cancels all jobs assigned to `user1` *or* `acct1`*.*
	`> mjobctl -c moab.48655 -w state=IDLE`
	Cancels job moab.48655, if it is idle.

-C - Checkpoint	
Format	JOBEXP
Description	Checkpoint a job. See 9.4 Checkpoint/Restart Facilities for more information.
Example	`> mjobctl -C job1045` *Checkpoint job* `job1045`*.*

-e - Rerun	
Format	JOBID
Description	Rerun the completed Torque job. This works only for jobs that are completed and show up in Torque as completed. This flag does not work with other resource managers.
Example	`> mjobctl -e job1045` *Rerun job* `job1045`*.*

-F - Force Cancel	
Format	JOBEXP

-F - Force Cancel	
Description	Forces a job to cancel and ignores previous cancellation attempts. Specifying this option tells Moab to purge a job from Torque (equivalent to qdel -p). This only tells pbs_server to remove any knowledge of the job from its internal memory. If the job is actually running, this will not cause pbs_server to tell the nodes with the job to cancel it. Therefore, users and admins should only use this form of mjobctl when they've confirmed that the job no longer exists on any compute nodes, and want to force Torque to stop tracking the job.
Example	`> mjobctl -F job1045` *Force cancel job `job1045`.*

-h - Hold	

Format	`<HOLDTYPE><JOBEXP>`
	
	`<HOLDTYPE>` = { user | batch | system | defer | ALL }

Default	user
Description	Set or release a job hold. See 9.1 Job Holds for more information.

Example	`> mjobctl -h user job1045`
	
	Set a user hold on job `job1045`.
	
	`> mjobctl -u all job1045`
	
	Unset all holds on job `job1045`.

-m - Modify	
Format	`<ATTR>{ += | =| -= } <VAL>` When using *`mjobctl -m`* with the hostlist attribute, only "=" is supported. If using Torque and *`mjobctl -m`* with the partition attribute, only "=" is supported. "+=", "-=", and "=" are supported with other resource managers (Native).

-m - Modify	
	`<ATTR>`={ account | advres | arraylimit | awduration| class | cpuclock | deadline | depend | eeduration | env | features | feature | flags | gres | group | hold | hostlist | jobdisk | jobmem | jobname | jobswap | loglevel | maxmem | messages | minstarttime | nodeaccess | nodecount | notificationaddress | partition | priority | queue | qos | reqreservation | rmxstring | reqattr | reqawduration | sysprio | tpn | trig | trigvar | user | userprio | var | wclimit}
Description	Modify a specific job attribute. If an `mjobctl -m` attribute can affect how a job starts, then it generally cannot affect a job that is already running. For example, it is not feasible to change the `hostlist` of a job that is already running. The userprio attribute enables you to specify user priority. For job priority, use the '-p' flag. Modification of the job dependency is also communicated to the resource manager in the case of PBS/Torque. Adding `--flags=warnifcompleted` causes a warning message to print when a job completes. To define values for `awduration`, `eeduration`, `minstarttime` (Note that the `minstarttime` attribute performs the same function as msub -a.), `reqawduration`, and `wclimit`, use the time spec format. A *non-active* job's partition list can be modified. If using Torque, only "=" (set) is supported. If using a Native Resource Manager you can add or subtract partitions, even multiple partitions. When adding or subtracting *multiple* partitions, each partition must have its own `-m partition{+= | = | -=}name` on the command line. An example for adding multiple partitions is provided in the list of examples. To modify a job's generic resources, use the following format: `gres{ += | = | -= } <gresName>[:<count>]`. `<gresName>` is a single resource, not a list. `<count>` is an integer that, if not specified, is assumed to be 1. Modifying a job's generic resources causes Moab to append the new gres (+=), subtract the specified gres (-=), or clear out all existing generic resources attached to the job and override them with the newly-specified one (=). If `<gresName>` is an empty string, all generic resources will be removed from the job. To modify the node access policy for a *queued* job, use `nodeaccess=[<policy>]`. See 4.2 Node Allocation Policies for a list of supported node access policies.

Example	`> mjobctl -m messages+="Adding a message" --flags=completed 1664`
	
	Set the message on the job, even if the job is completed.
	
	`> mjobctl -m reqawduration+=600 1664`

-m - Modify	

	Add 10 minutes to the job walltime.
	`> mjobctl -m eeduration=-1 1664`
	Reset job's effective queue time, to when the job was submitted.
	`> mjobctl -m var=Flag1=TRUE 1664`
	Set the job variable `Flag1` to `TRUE`.
	`> mjobctl -m notificationaddress="name@server.com"`
	Sets the notification email address associated with a job to `name@server.com`.
	`> mjobctl -m partition+=p3 -m partition+=p4 Moab.5`
	Adds multiple partitions (p3 and p4) to job `Moab.5`.
	Torque only supports "=" . "+=", "-=", and "=" are supported with other resource managers (Native).
	`> mjobctl -m arraylimit=10 sim.25`
	Changes the concurrently running subjob limit to 10 for array `sim.25`.
	`> mjobctl -m gres=matlab:1 job0201`
	Overrides all generic resources applied to job `job0201` and replaces them with 1 `matlab`.
	`> mjobctl -m user=user.job`
	Modifies the user of a job that was submitted directly to moab (msub) and has not yet been migrated.
	`> mjobctl -m userprio-=100 Moab.4`
	Reduces the user priority of `Moab.4` by 100.
	`> mjobctl -m tpn=2 Moab.128`
	Changes the requested "tasks per node" for job Moab.128 to 2.
	`> mjobctl -m maxmem=80mb 157`
	Modifies the total job memory of job 157. See the MAXMEM job extension for more information.

-N - Notify	
Format	[signal=]<SIGID>JOBEXP
Description	Send a signal to all jobs matching the job expression.

Example	`> mjobctl -N INT 1664`
	Send an interrupt signal to job 1664.
	`> mjobctl -N 47 1664`
	Send signal 47 to job `1664`.

-p - Priority	
Format	[+|+=|-=]<VAL><JOBID> [--flags=relative]
Description	Modify a job's system priority.

Example	Priority is the job priority plus the system priority. Each format affects the job and system priorities differently. Using the format `<VAL><JOBID>` or `+<VAL><JOBID>` will set the system priority to the maximum system priority plus the specified value. Using `+=<VAL><JOBID>` or `<VAL><JOBID> --flags=relative` will relatively increase the job's priority and set the system priority. Using the format `-=<VAL> <JOBID>` sets the system priority to 0, and does not change priority based on `<VAL>` (it will not decrease priority by that number).
	For the following example, `job1045` has a priority of 10, which is composed of a job priority of 10 and a system priority of 0.
	`> mjobctl -p +1000 job1045`
	The system priority changes to the max system priority plus `1000` points, ensuring that this job will be higher priority than all normal jobs. In this case, the job priority of 10 is not added, so the priority of `job1045` is now 1000001000.
	`> mjobctl -p -=1 job1045`
	The system priority of `job1045` resets to 0. The job priority is still 10, so the overall priority becomes 10.
	`> mjobctl -p 3 job1045 --flags=relative`
	Adds 3 points to the relative system priority. The priority for `job1045` changes from 10 to 13.

-q - Query	
Format	[diag(ALL)| hostlist | starttime| template] `<JOBEXP>`
Description	Query a job.

Example	`> mjobctl -q diag job1045`
	Query job `job1045`.
	`> mjobctl -q diag ALL --format=xml`
	Query all jobs and return the output in machine-readable XML.
	`> mjobctl -q starttime job1045`
	Query the estimated starttime of job `job1045`. The method used to estimate the start time can be specified by adding the --flags option with a value of prio, rsv, hist or all. If the --flags option is not specified, the value of the DEFAULTSTARTTIMEQUERY parameter will determine the default estimation method to be used -- which defaults to PRIORITY.
	`> mjobctl -q template <job>`
	Query job templates. If the `<job>` is set to ALL or empty, it will return information for all job templates.
	`> mjobctl -q wiki <jobName>`
	Query a job with the output displayed in a WIKI string. The job's name can be replaced with `ALL`.
	`--flags=completed` will only work with the `diag` option.

-r - Resume	
Format	JOBEXP
Description	Resume a job.
Example	`> mjobctl -r job1045` *Resume job `job1045`.*

-R - Requeue	
Format	JOBEXP [--flags=force|unmigrate]
Description	Requeue a job. Adding `--flags=force` forces an asynchronous requeue on Torque systems. Adding `--flags=unmigrate` causes Moab to pull a grid job back to the central scheduler for further evaluation on all valid partitions.
Example	`> mjobctl -R job1045` *Requeue job `job1045`.*

-s - Suspend	
Format	JOBEXP
Description	Suspend a job. For more information, see 9.3 Suspend/Resume Handling.
Example	`> mjobctl -s job1045` *Suspend job `job1045`.*

-u - Unhold	
Format	[<TYPE>[,<TYPE>]]JOBEXP <TYPE> = [user | system | batch | defer | ALL]
Default	`ALL`
Description	Release a hold on a job. See 9.1 Job Holds for more information.
Example	`> mjobctl -u user,system scrib.1045` *Release user and system holds on job `scrib.1045`.*

-w - Where	
Format	[CompletionTime | StartTime][<= | = | >=]<EPOCH_TIME>

-w - Where	
Description	Add a where constraint clause to the current command. As it pertains to `CompletionTime` | `StartTime`, the where constraint only works for completed jobs. CompletionTime filters according to the completed jobs' completion times; StartTime filters according to the completed jobs' start times.
Example	`> mjobctl -q diag ALL --flags=COMPLETED --format=xml -w CompletionTime>=1246428000 -w CompletionTime<=1254376800` *Prints all completed jobs still in memory that completed between July 1, 2024 and October 1, 2024.*

-x - Execute	
Format	JOBEXP
Description	Execute a job. The `-w` option allows flags to be set for the job. Allowable flags are, `ignorepolicies`, `ignorenodestate`, and `ignorersv`.

Example	`> mjobctl -x job1045`
	Execute job `job1045`.
	`> mjobctl -x -w flags=ignorepolicies job1046`
	Execute job `job1046` and ignore policies, such as MaxJobPerUser.

3.7.20.D Parameters

JOB EXPRESSION	
Format	`<STRING>`
Descriptio n	The name of a job or a regular expression for several jobs. The flags that support job expressions can use node expression syntax as described in 10.2.4 Node Selection. Using `x:` indicates the following string is to be interpreted as a regular expression, and using `r:` indicates the following string is to be interpreted as a range. Job expressions do not work for array subjobs. Moab uses regular expressions conforming to the POSIX 1003.2 standard. This standard is somewhat different than the regular expressions commonly used for filename matching in UNIX environments (see `man 7 regex`). To interpret a job expression as a regular expression, use `x:`.

JOB EXPRESSION	
	In most cases, it is necessary to quote the job expression (for example, `job13[5-9]`) to prevent the shell from intercepting and interpreting the special characters. The *`mjobctl`* command accepts a comma-delimited list of job expressions. Example usage might be `mjobctl -r job[1-2],job4` or `mjobctl -c job1,job2,job4`.

Example	`> mjobctl -c "x:80.*"`
	`job '802' canceled`
	`job '803' canceled`
	`job '804' canceled`
	`job '805' canceled`
	`job '806' canceled`
	`job '807' canceled`
	`job '808' canceled`
	`job '809' canceled`
	
	Cancel all jobs starting with `80`.
	
	`> mjobctl -m priority+=200 "x:74[3-5]"`
	`job '743' system priority modified`
	`job '744' system priority modified`
	`job '745' system priority modified`
	
	`> mjobctl -h x:17.*`
	`# This puts a hold on any job that has a 17 that is followed by an unlimited amount of any`
	`# character and includes jobs 1701, 17mjk10, and 17DjN_JW-07`
	
	`> mjobctl -h r:1-17`
	`# This puts a hold on jobs 1 through 17.`

3.7.20.E XML Output

`mjobctl` information can be reported as XML also. This is done with the command `mjobctl -q diag <JOB_ID>`.

XML Attributes

Name	Description
Account	The account assigned to the job.
AllocNodeList	The nodes allocated to the job.

Name	Description
Args	The job's executable arguments.
AWDuration	The active wall time consumed.
BlockReason	The block message index for the reason the job is not eligible.
Bypass	Number of times the job has been bypassed by other jobs.
Calendar	The job's timeframe constraint calendar.
Class	The class assigned to the job.
CmdFile	The command file path.
CompletionCode	The return code of the job as extracted from the resource manager.
CompletionTime	The time of the job's completion.
Cost	The cost of executing the job relative to an accounting manager.
CPULimit	The CPU limit for the job.
Depend	Any dependencies on the status of other jobs.
DRM	The master destination resource manager.
DRMJID	The master destination resource manager job ID.
EEDuration	The duration of time the job has been eligible for scheduling.
EFile	The stderr file.
Env	The job's environment variables set for execution.
EnvOverride	The job's overriding environment variables set for execution.
EState	The expected state of the job.
EstHistStartTime	The estimated historical start time.

Name	Description
EstPrioStartTime	The estimated priority start time.
EstRsvStartTime	The estimated reservation start time.
ExcHList	The excluded host list.
Flags	Command delimited list of Moab flags on the job.
GAttr	The requested generic attributes.
GJID	The global job ID.
Group	The group assigned to the job.
Hold	The hold list.
Holdtime	The time the job was put on hold.
HopCount	The hop count between the job's peers.
HostList	The requested host list.
IFlags	The internal flags for the job.
IsInteractive	If set, the job is interactive.
IsRestartable	If set, the job is restartable.
IsSuspendable	If set, the job is suspendable.
IWD	The directory where the job is executed.
JobID	The job's batch ID.
JobName	The user-specified name for the job.
JobGroup	The job ID relative to its group.
LogLevel	The individual log level for the job.

Name	Description
MasterHost	The specified host to run primary tasks on.
Messages	Any messages reported by Moab regarding the job.
MinPreemptTime	The minimum amount of time the job must run before being eligible for preemption.
Notification	Any events generated to notify the job's user.
OFile	The stdout file.
OldMessages	Any messages reported by Moab in the old message style regarding the job.
OWCLimit	The original wallclock limit.
PAL	The partition access list relative to the job.
QueueStatus	The job's queue status as generated this iteration.
QOS	The QoS assigned to the job.
QOSReq	The requested QoS for the job.
ReqAWDuration	The requested active walltime duration.
ReqCMaxTime	The requested latest allowed completion time.
ReqMem	The total memory requested/dedicated to the job.
ReqNodes	The number of requested nodes for the job.
ReqProcs	The number of requested procs for the job.
ReqReservation	The required reservation for the job.
ReqRMType	The required resource manager type.
ReqSMinTime	The requested earliest start time.
RM	The master source resource manager.

Name	Description
RMXString	The resource manager extension string.
RsvAccess	The list of reservations accessible by the job.
RsvStartTime	The reservation start time.
RunPriority	The effective job priority.
Shell	The execution shell's output.
SID	The job's system ID (parent cluster).
Size	The job's computational size.
STotCPU	The average CPU load tracked across all nodes.
SMaxCPU	The max CPU load tracked across all nodes.
STotMem	The average memory usage tracked across all nodes.
SMaxMem	The max memory usage tracked across all nodes.
SRMJID	The source resource manager's ID for the job.
StartCount	The number of the times the job has tried to start.
StartPriority	The effective job priority.
StartTime	The most recent time the job started executing.
State	The state of the job as reported by Moab.
StatMSUtl	The total number of memory seconds utilized.
StatPSDed	The total number of processor seconds dedicated to the job.
StatPSUtl	The total number of processor seconds utilized by the job.
StdErr	The path to the stderr file.

Name	Description
StdIn	The path to the stdin file.
StdOut	The path to the stdout file.
StepID	StepID of the job (used with LoadLeveler systems).
SubmitHost	The host where the job was submitted.
SubmitLanguage	The resource manager language that the submission request was performed.
SubmitString	The string containing the entire submission request.
SubmissionTime	The time the job was submitted.
SuspendDuration	The amount of time the job has been suspended.
SysPrio	The admin specified job priority.
SysSMinTime	The system specified min. start time.
TaskMap	The allocation taskmap for the job.
TermTime	The time the job was terminated.
User	The user assigned to the job.
UserPrio	The user specified job priority.
UtlMem	The utilized memory of the job.
UtlProcs	The number of utilized processors by the job.
Variable	
VWCTime	The virtual wallclock limit.

3.7.20.F Example

```
> mjobctl -q diag ALL --format=xml
<Data><job AWDuration="346" Class="batch" CmdFile="jobsleep.sh" EEDuration="0"
```

```
EState="Running" Flags="RESTARTABLE" Group="test" IWD="/home/test" JobID="11578"
QOS="high"
RMJID="11578.lolo.icluster.org" ReqAWDuration="00:10:00" ReqNodes="1" ReqProcs="1"
StartCount="1"
StartPriority="1" StartTime="1083861225" StatMSUtl="903.570" StatPSDed="364.610"
StatPSUtl="364.610"
State="Running" SubmissionTime="1083861225" SuspendDuration="0" SysPrio="0"
SysSMinTime="00:00:00"
User="test"><req AllocNodeList="hana" AllocPartition="access" ReqNodeFeature="[NONE]"
ReqPartition="access"></req></job><job AWDuration="346" Class="batch"
CmdFile="jobsleep.sh"
EEDuration="0" EState="Running" Flags="RESTARTABLE" Group="test" IWD="/home/test"
JobID="11579"
QOS="high" RMJID="11579.lolo.icluster.org" ReqAWDuration="00:10:00" ReqNodes="1"
ReqProcs="1"
StartCount="1" StartPriority="1" StartTime="1083861225" StatMSUtl="602.380"
StatPSDed="364.610"
StatPSUtl="364.610" State="Running" SubmissionTime="1083861225" SuspendDuration="0"
SysPrio="0"
SysSMinTime="00:00:00" User="test"><req AllocNodeList="lolo" AllocPartition="access"
ReqNodeFeature="[NONE]" ReqPartition="access"></req></job></Data>
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.38.K setspri
- 3.7.38.A canceljob
- 3.7.38.G runjob

3.7.21 mnodectl

3.7.21.A Synopsis

mnodectl -m attr{=|-=|+=}val nodeexp
mnodectl -q [cat|diag|profile|wiki] nodeexp

[--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]

3.7.21.B Overview

Change specified attributes for a given node expression.

3.7.21.C Access

By default, this command can be run by any Moab Admin.

3.7.21.D Options

-m - Modify	
Format	`<ATTR>{=|-=|+=}<VAL>` Where `<ATTR>` is one of the following: CFGCLASS, FEATURES, GEVENT, GMETRIC, MESSAGE, OS, POWER, STATE, or VARIABLE and -=, except when used for features, clears the attribute instead of decrementing the attribute's value and = indicates that you are specifying a new value to replace the old one(s), if any. When the -= option is used to modify features, it removes the specified features from the node. The += option, which is only available for features, enables you to append additional features to the current list rather than replacing the current list entirely. Changing OS and POWER require a Moab Adaptive Computing Suite license and a provisioning resource manager.
Description	Modify the state or attribute of specified node(s).

Example	`> mnodectl -m cfgclass-=debug node1`
	`> mnodectl -m features+=fastio,highmem node1`
	`> mnodectl -m gevent=cpufail:'cpu02 has failed w/ec:0317' node1`
	`> mnodectl -m gmetric=temp:131.2 node1`
	`> mnodectl -m message='cpufailure:cpu02 has failed w/ec:0317' node1`
	`> mnodectl -m OS=RHAS30 node1`
	`> mnodectl -m power=off node1`
	`> mnodectl -m state=idle node1`
	`> mnodectl -m variable=IP=10.10.10.100,Location=R1S2 node1`

-q - Query	
Format	{cat | diag | profile | wiki}
Description	Query node categories or node profile information (see the ENABLEPROFILING node attribute for nodes). The `diag` and `profile` options must use `--xml`.
Example	

-q - Query	

	```
	> mnodectl -q cat ALL
	node categorization stats from Mon Jul 10 00:00:00 to Mon Jul 10 15:30:00
	Node: moab
	 Categories:
	                    busy: 96.88%
	                    idle: 3.12%
	Node: maka
	 Categories:
	                    busy: 96.88%
	                    idle: 3.12%
	Node: pau
	 Categories:
	                    busy: 96.88%
	                    idle: 3.12%
	Node: maowu
	 Categories:
	                    busy: 96.88%
	                 down-hw: 3.12%
	Cluster Summary:
	                    busy: 96.88%
	                 down-hw: 0.78%
	                    idle: 2.34%
	```
	```
	> mnodectl -v -q profile
	...
	```
	```
	> mnodectl -q wiki <ALL>
	GLOBAL STATE=Idle PARTITION=SHARED
	n0 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n1 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n2 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n3 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n4 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n5 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n6 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n7 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n8 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	n9 STATE=Idle PARTITION=base APROC=4 CPROC=4 RM=base NODEACCESSPOLICY=SHARED
	```
	Query a node with the output displayed in a WIKI string.

Parameters

CFGCLASS	
Format	`<STRING>`
Description	Class name. Only "-=" is supported when modifying cfgclass on a node. To add or set classes on a node, see the attribute HOSTLIST
Example	`> mnodectl -m cfgclass-=debug node1`

FEATURES	
Format	`<STRING>` One of the following: • a comma-delimited list of features • [NONE] (to clear features on the node)
Description	Sets the features on a node. These node features will be overwritten when a resource manager reports features.

Example	`mnodectl -m features=fastio,highmem node1`
	`mnodectl -m features=[NONE] node1`

GEVENT	
Format	`<EVENT>:<MESSAGE>`
Description	Creates a generic event on the node to which Moab can respond (see 10.9 Enabling Generic Events).
Example	`mnodectl -m gevent=powerfail:'power has failed' node1`

GMETRIC	
Format	`<ATTR>:<VALUE>`
Description	Sets the value for a generic metric on the node (see 10.9 Enabling Generic Events). When a gmetric set in Moab conflicts with what the resource manager reports, Moab uses the set gmetric until the next time the resource manager reports a different number.
Example	`mnodectl -m gmetric=temp:120 node1`

MESSAGE	
Format	`'<MESSAGE>'`

MESSAGE	
Description	Sets a message to be displayed on the node.
Example	`mnodectl -m message='powerfailure: power has failed' node1`

NODEEXP	
Format	`<STRING>` Where `<NODEEXP>` is a node name, regex or ALL. Node regex has the potential to unintentionally match many nodes (for example, specifying n1 will match n10, n11, n12, n100, etc). To ensure correct matching, explicitly use the "x:<node_regex>" when modifying multiple nodes in one command. Currently, this is supported for features.
Description	Identifies one or more nodes.

Example	`node1` - applies only to `node1`
	`fr10n*` - all nodes starting with `fr10n`
	`ALL` - all known nodes

OS	
Format	`<STRING>`
Description	Operating System (see 11.8 Resource Provisioning).
Example	`mnodectl node1 -m OS=RHELAS30`

POWER	
Format	{off|on}
Description	Set the power state of a node. Action will NOT be taken if the node is already in the specified state. If you power off a node, a green policy will try to turn it back on. If you want the node to remain powered off, you must associate a reservation with it.

POWER	
	If you request to power off a node that has active work on it, Moab returns a status indicating that the node is busy with a job and will not be powered off. You will see one of these messages: • `Ignoring node <name>: power ON in process` (indicates node is currently powering on) • `Ignoring node <name>: power OFF in process` (indicates node is currently powering off) • `Ignoring node <name>: has active jobs running` (indicates the node is currently running active jobs) Once you resolve the activity on the node (by preempting or migrating the jobs, for example), you can attempt to power the node off again. You can use the `--flags=force` option to cause a force override. However, doing this will power off the node regardless of whether or not its jobs get migrated or preempted (i.e., you run the risk of losing the jobs entirely). For example: `> mnodectl node1 -m power=off --flags=force`
Example	`> mnodectl node1 -m power=off`

STATE	
Format	{drained|idle}
Description	Remove (drained) or add (idle) a node from scheduling.
Example	`mnodectl node1 -m state=drained` *Moab ignores `node1` when scheduling.*

VARIABLE	
Format	`<name>`[=`<value>`],`<name>`[=`<value>`]...
Description	Set a list of variables for a node.
Example	`> mnodectl node1 -m variable=IP=10.10.10.100,Location=R1S2`

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.10 mdiag -n
- 3.7.33 showres
- 3.7.2 checknode
- -n[<NODEID>] - reports current and historical node statistics

3.7.22 moab

3.7.22.A Synopsis

`moab` --about --help --loglevel=<LOGLEVEL> --version [-c <CONFIG_FILE>] [-C] [-d] [-e] [-h] [-P [<PAUSEDURATION>]] [-R <RECYCLEDURATION>] [-s] [-S [<STOPITERATION>]] [-v]

3.7.22.B Options

Option	Description
--about	Displays build environment and version information.
--loglevel	Sets the server loglevel to the specified value.
--version	Displays version information.
-c	Configuration file the server should use.
-C	Clears checkpoint files (`.moab.ck`, `.moab.ck.1`).
-d	Debug mode (does not background itself).
-e	Forces Moab to exit if there are any errors in the configuration file, if it can't connect to the configured database, or if it can't find these directories: • `statdir` • `logdir` • `spooldir`

Option	Description
	• `toolsdir`
-P	Starts Moab in a paused state for the duration specified (default: pause indefinitely; resume with `mschedctl -r` (or `-R`), or a service restart).
-R	Causes Moab to automatically recycle every time the specified duration transpires.
-s	Starts Moab in the state that was most recently checkpointed.
-S	Suspends/stops scheduling at specified iteration (or at startup if no iteration is specified).
-v	Same as --version.

3.7.23 mrmctl

3.7.23.A Synopsis

mrmctl -f [`<fobject>`] {`<rmid>` | AM[:`<amid>`] | ID[:`<imid>`]}

mrmctl -l [`<rmid>` | AM[:`<amid>`]]

mrmctl -m <attr>=<value> [`<rmid>`]

mrmctl -p {`<rmid>` | AM[:`<amid>`]}

mrmctl -q AccountBalanceCache AM[:`<amid>`]

mrmctl -R {AM[:`<amid>`] | ID[:`<imid>`]}

```
[--about] [--help] [--host=<serverHostName>] [--
loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>]
[--timeout=<seconds>] [--version] [--xml]
```

3.7.23.B Overview

`mrmctl` allows an admin to query, list, modify, and ping the resource managers and accounting managers in Moab. *`mrmctl`* also allows for a queue (often referred to as a class) to be created for a resource manager.

3.7.23.C Access

By default, this command can be run by level 1 and level 2 Moab Admins (see the parameter ADMINCFG).

3.7.23.D Options

-f	
Format	-f [`<fobject>`] where fobject is optional and one of messages or stats.
Default	If no `fobject` is specified, then reported failures and performance data will be flushed. If no resource manager ID is specified, the first resource manager will be flushed.
Description	Clears resource manager statistics. If messages is specified, then reported failures, performance data, and messages will be flushed.
Example	`> mrmctl -f base` *Moab will clear the statistics for resource manager `base`.*

-l	
Format	-l
Default	All resource managers and accounting managers (when no resource manager / accounting manager is specified).
Description	List Resource and Accounting Manager(s).
Example	`> mrmctl -l` *Moab will list all resource managers and accounting managers.*

-m	
Format	-m <attr>=<val>
Default	All resource managers and accounting managers (when no resource manager / accounting manager is specified).

-m	
Description	Modify resource managers and accounting managers.
Example	`> mrmctl -m state=disabled peer13`

-p	
Format	-p
Default	First resource manager configured.
Description	Ping resource manager.
Example	`> mrmctl -p base` *Moab will ping resource manager* `base`*.*

-q	
Format	-q AccountBalanceCache
Default	---
Description	When an accounting manager is being used and the fast-allocation accounting mode is configured, this option queries Moab's internal cache of account balances. See 5.5.2 Accounting Mode. Also see 'Select an Appropriate Accounting Mode' in the *Moab Accounting Manager Administrator Guide* for more information.
Example	`> mrmctl -q AccountBalanceCache AM`

-R	
Format	-R
Description	Dynamically reloads server information for the identity manager service if `ID` is specified; if `AM` is specified, reloads the accounting manager service.
Example	`> mrmctl -R ID`

-R	
	Reloads the identity manager on demand.

Resource manager interfaces can be enabled/disabled using the modify operation to change the resource manager state as in the following example:

```
# disable active resource manager interface
> mrmctl -m state=disabled torque
# restore disabled resource manager interface
> mrmctl -m state=enabled torque
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.14 mdiag -R
- 3.7.7 mdiag -c

3.7.24 mrsvctl

3.7.24.A Synopsis

mrsvctl -B*SRSVID*

mrsvctl -c [-a *ACL*] [-b *SUBTYPE*] [-d *DURATION*] [-D *DESCRIPTION*] [-e *ENDTIME*] [-E *EXCLUSIVE*] [-f *FEATURES*] [-F *FLAGS*] [-g *RSVGROUP*] [-h *HOSTLIST*] [-n *NAME*] [-o *OWNER*] [-p *PARTITION*] [-P *PROFILE*] [-R *RESOURCES*] [-s *STARTTIME*] [-S *SET ATTRIBUTE*] [-t *TASKS*] [-T *TRIGGER*] [-V *VARIABLE*] [-x *JOBLIST*]

mrsvctl -C [`-g` *SRSVID*] {*RESERVATION PATTERN*}

mrsvctl -l [{*RESERVATION PATTERN* | -i *INDEX*}]

mrsvctl -m `<duration|endtime|hostexp|loglevel|reqtaskcount|rsvaccesslist|rsvgroup|starttime|variable>{=|+=|-=}`*<VAL> <hostexp>*{`+=|-=`}*<VAL> <variable>*{+=*KEY*=*VAL*|-=*KEY_TO_REMOVE*} {*RESERVATION PATTERN* | -i *INDEX*}

mrsvctl -q {*RESERVATION PATTERN* | -i *INDEX*} [`--blocking`]

mrsvctl -r {*RESERVATION PATTERN* | -i *INDEX*}

```
[--about] [--help] [--host=<serverHostName>] [--
loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>]
[--timeout=<seconds>] [--version] [--xml]
```

3.7.24.B Overview

mrsvctl controls the creation, modification, querying, and releasing of reservations.

The timeframe covered by the reservation can be specified on either an absolute or relative basis. Only jobs with credentials listed in the reservation's access control list can utilize the reserved resources. However, these jobs still have the freedom to utilize resources outside of the reservation. The reservation will be assigned a name derived from the ACL specified. If no reservation ACL is specified, the reservation is created as a system reservation and no jobs will be allowed access to the resources during the specified timeframe (valuable for system maintenance, etc.). See 6.1.1 Reservation Overview for more information.

Reservations can be viewed using the `-q` flag and can be released using the `-r` flag.

By default, reservations are not exclusive and can overlap with other reservations and jobs. Use the '-E' flag to adjust this behavior.

3.7.24.C Access

By default, this command can be run by level 1 and level 2 Moab Admins (see the parameter ADMINCFG).

3.7.24.D Options

-a	
Name	ACL
Format	`<TYPE>==<VAL>[,<TYPE>==<VAL>]...` Where `<TYPE>` is one of the following: ACCT, CLASS, DURATION, GROUP, JATTR, PROC, QOS, USER
Description	List of limitations for access to the reserved resources (see also the section ACL Modifiers).
Example	

-a	

	`> mrsvctl -c -h node01 -a USER==john+,CLASS==batch-`
	Moab will make a reservation on `node01` allowing access to user john and restricting access from class batch when other resources are available to class batch.
	
	`> mrsvctl -m -a USER-=john system.1`
	Moab will remove user john from the `system.1` reservation.

Notes	• When you specify multiple credentials, a user must only match one of them in order to access the reservation. To require one or more of the listed limitations for reservation access, each required specification must end with an asterisk (*). If a user meets the required limitation(s), that user has access to the reservation (without meeting any that are not marked required). • There are three different assignment operators that can be used for modifying most credentials in the ACL. The operator == will reassess the list for that particular credential type. The += operator will append to the list for that credential type, and -= will remove from the list. Two other operators are used to specify DURATION and PROC: >= (greater than) and <= (less than). • To add multiple credentials of the same type with one command, use a colon to separate them. To separate lists of different credential types, use commas. For example, to reassign the user list to consist of users Joe and Bob, and to append the group MyGroup to the groups list on the `system.1` reservation, you could use the command `mrsvctl -m -a USER==Joe:Bob,GROUP+=MyGroup system.1`. • Any of the ACL modifiers can be used. When using them, it is often useful to put single quotes on either side of the assignment command. For example, `mrsvctl -m -a 'USER==&Joe' system.1`. • Some flags are mutually exclusive. For example, the ! modifier means that the credential is blocked from the reservation and the & modifier means that the credential must run on that reservation. Moab will take the most recently parsed modifier. Modifiers can be placed on either the left or the right of the argument, so `USER==&JOE` and `USER==JOE&` are equivalent. Moab parses each argument starting from right to left on the right side of the argument, then from left to right on the left side. So, if the command was `USER==!Joe&`, Moab keeps the equivalent of `USER==!Joe` because the `!` is the last one parsed. • You can set a reservation to have a time limit for submitted jobs using DURATION and the * modifier. For example, `mrsvctl -m -a 'DURATION<=*1:00:00' system.1` causes the `system.1` reservation to not accept any jobs with a walltime greater than one hour.

-a	

	Similarly, you can set a reservation to have a processor limit using PROC and the `*` modifier. `mrsvctl -a 'PROC>=2*' system.2` causes the `system.2` reservation to only allow jobs requesting more than 2 procs to run on it.
	• You can verify the ACL of a reservation using the *`mdiag -r`* command.
	`mrsvctl -m -a 'USER==Joe:Bob,GROUP-=BadGroup,ACCT+=GoodAccount,DURATION<= *1:00:00' system.1`
	Moab will reassign the `USER` list to be `Joe` and `Bob`, will remove `BadGroup` from the `GROUP` list, append `GoodAccount` to the `ACCT` list, and only allow jobs that have a submitted walltime of an hour or less on the `system.1` reservation.
	`mrsvctl -m -a 'USER==Joe,USER==Bob' system.1`
	Moab will assign the `USER` list to `Joe`, and then reassign it again to `Bob`. The final result will be that the `USER` list will just be `Bob`. To add `Joe` and `Bob`, use `mrsvctl -m -a USER==Joe:Bob system.1` or `mrsvctl -m -a USER==Joe,USER+=Bob system.1`.

-b	
Name	SUBTYPE
Format	One of the node category values or node category shortcuts.
Description	Add subtype to reservation.
Example	`> mrsvctl -c -b SoftwareMaintenance -t ALL` *Moab will associate the reserved nodes with the node category SoftwareMaintenance.*

-B	
Name	REBUILD
Format	`<SRSVID>`
Description	Rebuilds standing reservations while Moab is running.

-B	
Example	`> mrsvctl -B <SRSVID>`

-c	
Name	CREATE
Format	`<ARGUMENTS>`
Description	Creates a reservation. If a created reservation has a given duration but the start time is in the past, one of the following actions occur depending on whether the present time falls within the reservation's given duration: • If the present time is still within the reservation's duration time frame, the start time does not change and the reservation shows however long is left in the reservation (present time minus the duration time). • If present time is outside of the reservation's duration time frame, the reservation start time automatically sets to the present time and the reservation continues for its full given duration. The `-x` flag, when used with `-F ignjobrsv`, lets users create reservations but exclude certain nodes from being part of the reservation because they are running specific jobs. The `-F` flag instructs *`mrsvctl`* to still consider nodes with current running jobs.

Examples	`> mrsvctl -c -t ALL`
	Moab will create a reservation across all system resources.
	`> mrsvctl -c -t 5 -F ignjobrsv -x moab.5,moab.6`
	Moab will create the reservation while assigning the nodes. Nodes running jobs `moab5` and `moab6` will not be assigned to the reservation.
	`> mrsvctl -c -t 1 -d INFINITY`
	Moab will create an infinite reservation.

-C	
Name	CLEAR
Format	`<RSVID> | -g <SRSVID>`
Description	Clears any disabled time slots from standing reservations and allows the recreation of disabled reservations.
Example	`> mrsvctl -C -g testing` *Moab will clear any disabled timeslots from the standing reservation* `testing`*.*

-d	
Name	DURATION
Format	`[[[DD:]HH:]MM:]SS`
Default	INFINITY
Description	Duration of the reservation (not needed if ENDTIME is specified).

Example	`> mrsvctl -c -h node01 -d 5:00:00`
	Moab will create a reservation on `node01` *lasting 5 hours.*
	`> mrsvctl -c -h node01 -d INFINITY`
	Moab will create a reservation with a duration of INFINITY (no endtime).

-D	
Name	DESCRIPTION
Format	`<STRING>`
Description	Human-readable description of reservation or purpose.
Example	`> mrsvctl -c -h node01 -d 5:00:00 -D 'system maintenance to test network'`

-D	
	Moab will create a reservation on `node01` lasting 5 hours.

-e	
Name	ENDTIME
Format	[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS
Default	INFINITY
Description	Absolute or relative time reservation will end (not required if Duration specified). ENDTIME also supports an epoch timestamp.
Example	`> mrsvctl -c -h node01 -e +3:00:00` *Moab will create a reservation on `node01` ending in 3 hours.*

-E	
Name	EXCLUSIVE
Description	When specified, Moab will only create a reservation if there are no other reservations (exclusive or otherwise) that would conflict with the time and space constraints of this reservation. If exceptions are desired, the rsvaccesslist attribute can be set or the ignrsv flag can be used.
Example	`> mrsvctl -c -h node01 -E` *Moab will only create a reservation on `node01` if no conflicting reservations are found.* This flag is only used at the time of reservation creation. Once the reservation is created, Moab allows jobs into the reservation based on the ACL. Also, once the exclusive reservation is created, it is possible that Moab will overlap it with jobs that match the ACL.

-f	
Name	FEATURES
Format	`<STRING>[:<STRING>]...`
Description	List of node features that must be possessed by the reserved resources. You can use a backslash and pipe to delimit features to indicate that Moab can use one or the other.
Example	`> mrsvctl -c -h node[0-9] -f fast|slow` *Moab will create a reservation on nodes matching the expression and which also have either the feature* `fast` *or the feature* `slow`*.*

-F	
Name	FLAGS
Format	`<flag>[[,<flag>]...]`
Description	Comma-delimited list of flags to set for the reservation (see the section Flags for flags).
Example	`> mrsvctl -c -h node01 -F ignstate` *Moab will create a reservation on* `node01` *ignoring any conflicting node states.*

-g	
Name	RSVGROUP
Format	`<STRING>`
Description	For a create operation, create a reservation in this reservation group. For list and modify operations, take actions on all reservations in the specified reservation group. The `-g` option can also be used in conjunction with the -r option to release a reservation associated with a specified group. See 6.1.1.G Reservation Group for more information.
Example	

-g	
	`> mrsvctl -c -g staff -h 'node0[1-9]'` *Moab will create a reservation on nodes matching the node expression given and assign it to the reservation group `staff`.*

-h	
Name	HOSTLIST
Format	class:`<classname>`[,`<classname>`]... or `<STRING>` or 'r:`<nodeNameStart>`[`<beginRange>`-`<endRange>`]' or ALL
Description	Host expression or a class mapping indicating the nodes that the reservation will allocate. When you specify a `<STRING>`, the HOSTLIST attribute is always treated as a regular expression. `foo10` will map to `foo10`, `foo101`, `foo1006`, etc. To request an exact host match, the expression can be bounded by the carat and dollar op expression markers as in `^foo10$`.
Example	`> mrsvctl -c -h 'r:node0[1-9]'` *Moab will create a reservation on nodes `node01`, `node02`, `node03`, `node04`, `node05`, `node06`, `node07`, `node08`, and `node09`.* `> mrsvctl -c -h class:batch` *Moab will create a reservation on all nodes that support class/queue `batch`.*

-i	
Name	INDEX

-i	
Format	`<STRING>`
Description	Use the reservation index instead of full reservation ID.
Example	`> mrsvctl -m -i 1 starttime=+5:00` *Moab will create a reservation on nodes matching the expression given.*

-l	
Name	LIST
Format	`<RSV_ID>` or ALL `RSV_ID` can be the name of a reservation or a regular expression.
Default	ALL
Description	List reservation(s).
Example	`> mrsvctl -l system*` *Moab will list all of the reservations whose names start with* `system`.

-m	
Name	MODIFY

-m	
Format	<ATTR>=<VAL>[-m <ATTR2>=<VAL2>]... Where ATTR is one of the following: **flags** **duration**: duration{+=|-=|=}<RELTIME> **endtime**: endtime{+=|-=}<RELTIME> or endtime=<ABSTIME> **hostexp**: hostexp[+=|-=]<node>[,<node>] **loglevel**: loglevel[=]<loglevel> **reqtaskcount**: reqtaskcount{+=|-=|=}<TASKCOUNT> **rsvaccesslist**: rsvaccesslist[=]<reservation> **rsvgroup**: rsvgroup[=]<rsvgroup> **starttime**: starttime{+=|-=}<RELTIME> or starttime=<ABSTIME> **variable**: variable[+=key1=val1|-=key_to_remove]
Description	Modify aspects of a reservation. Moab is constantly scheduling and updating reservations. Before modifying a reservation we recommend that you first stop the scheduler (*mschedclt -s*) so that the scheduler and reservation are in a stable and steady state. Once the reservation has been modified, resume the scheduler with *mschedctl -r*.

Example	`> mrsvctl -m duration=2:00:00 system.1`
	
	Moab sets the duration of reservation system.1 to be exactly two hours, therefore modifying the endtime of the reservation.
	
	`> mrsvctl -m duration+=5:00:00 system.1`
	
	Moab extends the duration of system.1 by five hours.
	
	`> mrsvctl -m endtime+=5:00:00 system.1`

-m	
	Moab moves the endtime of reservation `system.1` ahead by five hours.

```
> mrsvctl -m endtime-=5:00:00 system.1
```

Moab moves the endtime of reservation `system.1` five hours from its current endtime (without modifying the starttime; therefore, this action is equivalent to modifying the duration of the reservation).

```
> mrsvctl -m hostexp+=node02 system.1
```

Moab adds node02 to the hostlist for reservation `system.1`.

```
> mrsvctl -m loglevel=2 system.1
```

Overrides the global LOGLEVEL parameter when dealing with events related to the reservation. LOGLEVEL values are 0-9, where 9 is most verbose.

```
> mrsvctl -m reqtaskcount+=5 system.1
```

Increases the TASKCOUNT for the `system.1` reservation by 5.

```
> mrsvctl -m rsvaccesslist=network system.1
```

Gives the `system.1` reservation access to the `network` reservation.

```
> mrsvctl -m rsvgroup=network system.1
```

Changes reservation `system.1` to the `network` RSVGROUP.

```
> mrsvctl -m starttime+=5:00:00 system.1
```

Moab advances the starttime of `system.1` five hours from its current starttime (without modifying the duration of the reservation).

```
> mrsvctl -m starttime=15:00:00_7/6/24 system.1
```

Moab sets the starttime of reservation `system.1` to 3:00 P.M. on July 6, 2024.

```
> mrsvctl -m starttime-=5:00:00 system.1
```

Moab moves the starttime of reservation `system.1` ahead five hours.

```
> mrsvctl -m variable+key1=val1 system.1
```

-m	

	Moab adds the variable `key1` with the value `key2` to `system.1`.
	`> mrsvctl -m variable+=key1=val1 variable+=key2=val2 system.1`
	Moab adds the variable `key1` with the value `val1`, and variable `key2` with `val2` to `system.1`. (Note that each variable flag requires a distinct `-m` entry.)
	`> mrsvctl -m variable-=key1 system.1`
	Moab deletes the variable `key1` from `system.1`.
	`> mrsvctl -m variable-=key1 -m variable-=key2 system.1`
	Moab deletes the variables `key1` and `key2` from `system.1`.

Notes:	• Modifying the starttime does not change the duration of the reservation, so the endtime changes as well. The starttime can be changed to be before the current time, but if the change causes the endtime to be before the current time, the change is not allowed.
	• Modifying the endtime changes the duration of the reservation as well (and vice versa). An endtime *cannot* be placed before the starttime or before the current time.
	• Duration cannot be negative.
	• The += and -= operators operate on the time of the reservation (`starttime+=5` adds five seconds to the current reservation starttime), while + and - operate on the current time (`starttime+5` sets the starttime to five seconds from now).
	• If the starttime or endtime specified is before the current time without a date specified, it is set to the next time that fits the command. To force the date, add the date as well. For the following examples, assume that the current time is 9:00 A.M. on March 1, 2025.
	`> mrsvctl -m starttime=8:00:00_3/1/25 system.1`
	Moab moves `system.1`'s starttime to 8:00 A.M., March 1.
	`> mrsvctl -m starttime=8:00:00 system.1`
	Moab moves `system.1`'s starttime to 8:00 A.M., March 2.
	`> mrsvctl -m endtime=7:00:00 system.1`
	Moab moves `system.1`'s endtime to 7:00 A.M., March 3. This happens because the endtime must also be after the starttime, so Moab continues searching until it has found a valid time that is in the future and after the starttime.

-m	

	```
	> mrsvctl -m endtime=7:00:00_3/2/25 system.1
	```
	Moab will return an error because the endtime cannot be before the starttime.

-n	
Name	NAME
Format	`<STRING>`
Description	Name for new reservation. If no name is specified, the reservation name is set to first name listed in ACL or `SYSTEM` if no ACL is specified. Reservation names cannot contain whitespace.

Example	
	```
	mrsvctl -c -h node01 -n John
	```
	Moab will create a reservation on node01 with the name `John`.

-o	
Name	OWNER
Format	`<CREDTYPE>:<CREDID>`
Description	The owner of a reservation. See the section Reservation Ownership for more information.

Example	
	```
	mrsvctl -c -h node01 -o USER:user1
	```
	Moab creates a reservation on `node01` owned by `user1`.

-p	
Name	PARTITION
Format	`<STRING>`
Description	Only allocate resources from the specified partition.
Example	`mrsvctl -c -p switchB -t 14` *Moab will allocate 14 tasks from the `switchB` partition.*

-P	
Name	PROFILE
Format	`<STRING>`
Description	Indicates the reservation profile to load when creating this reservation.
Example	`mrsvctl -c -P testing2 -t 14` *Moab will allocate 14 tasks to a reservation defined by the `testing2` reservation profile.*

-q	
Name	QUERY
Format	`<RSV_ID>` — The `-r` option accepts x: node regular expressions and r: node range expressions [asterisks (*) are supported wildcards also].
Description	Get diagnostic information or list all completed reservations. The command gathers information from the Moab cache, which prevents it from interrupting the scheduler, but the `--blocking` option can be used to bypass the cache and interrupt the scheduler.
Example	`mrsvctl -q ALL` *Moab will query reservations.*

-q	
	`mrsvctl -q system.1` *Moab will query the reservation `system.1`.*

-r	
Name	RELEASE
Format	`<RSV_ID>` — The `-r` option accepts x: node regular expressions and r: node range expressions [asterisks (`*`) are supported wildcards also].
Description	Releases the specified reservation. When you release an instance of a standing reservation, Moab will remember that and prevent a reservation from being created for that same period (even after a restart of Moab). When Moab reaches the end of the period, it will still create new reservations in the future to meet the reservation depth requirement.

Example	`> mrsvctl -r system.1`
	Moab will release reservation `system.1`.
	
	`> mrsvctl -r -g idle`
	Moab will release all idle job reservations.

-R	
Name	RESOURCES
Format	`<tid>` or `<RES>=<VAL>`[{,|+|;}`<RES>=<VAL>`]... Where `<RES>` is one of the following: PROCS, MEM, DISK, SWAP, GRES
Default	PROCS=-1

-R	
Description	The resources to be reserved per task (`-1` indicates all resources on node). When specifying multiple resources, enclose the resource list in single quotes and separate the resource identifiers with semicolons (example: 'MEM=100;PROCS=1'). Alternatively, you can omit the single quotes and separate the resource identifiers with escaped semicolons (example: MEM=100\;PROCS=1). For GRES resources, `<VAL>` is specified in the format `<GRESNAME>[:<COUNT>]`

Example	`> mrsvctl -c -R 'MEM=100;PROCS=2' -t 2`
	Moab will create a reservation for two tasks with the specified resources.
	`> mrsvctl -c -R GRES:licenseA:100 -t 1`
	Moab will reserve 100 instances of "licenseA".

-s	
Name	STARTTIME

Format	`[HH[:MM[:SS]]][_MO[/DD[/YY]]]`
	or
	`+[[[DD:]HH:]MM:]SS`

Default	[NOW]
Description	Absolute or relative time reservation will start. STARTTIME also supports an epoch timestamp.

Example	`> mrsvctl -c -t ALL -s 3:00:00_4/4/24`
	Moab will create a reservation on all system resources at 3:00 am on April 4, 2024.
	`> mrsvctl -c -h node01 -s +5:00`
	Moab will create a reservation in 5 minutes on `node01`.

-s	

	```
	> mrsvctl -m -s -=5:00 system.1
	```
	This will decrement the start time by 5 minutes.

-S	
Name	SET ATTRIBUTE
Format	`<ATTR>=<VALUE>`, where `<ATTR>` is one of the following: • aaccount - accountable account • agroup - accountable group • aqos - accountable QoS • auser - accountable user • reqarch - required architecture • reqmemory - required node memory (in MB) • reqos - required operating system • rsvaccesslist - comma-delimited list of reservations or reservation groups that can be accessed by this reservation request. Because each reservation can access all other reservations by default, you should make any reservation with a specified rsvaccesslist exclusive by setting the -E flag. This setting gives the otherwise exclusive reservation access to reservations specified in the list.
Description	A reservation attribute will be used to create this reservation.

Example	
	```
	> mrsvctl -c -h node01 -S aqos=high
	```
	Moab will create a reservation on `node01` and will use the QOS `high` as the accountable credential.

-t	
Name	TASKS
Format	`<INTEGER>`[-`<INTEGER>`]
Description	The number of tasks to reserve. ALL indicates all resources available should be reserved.

-t	
	If the task value is set to ALL, Moab applies the reservation regardless of existing reservations and exclusive issues. If an integer is used, Moab only allocates accessible resources. If a range is specified Moab attempts to reserve the maximum number of tasks, or at least the minimum.

Example	`> mrsvctl -c -t ALL`
	Moab will create a reservation on all resources.
	`> mrsvctl -c -t 3`
	Moab will create a reservation for three tasks.
	`> mrsvctl -c -t 3-10 -E`
	Moab will attempt to reserve 10 tasks but will fail if it cannot get at least three.

-T	
Name	TRIGGER
Format	`<STRING>`
Description	Comma-delimited reservation trigger list following format described in the trigger format section of the reservation configuration overview. See 17.2.1 Creating a Trigger for more information. To cancel a standing reservation with a trigger, the `SRCFG` parameter's attribute DEPTH must be set to `0`.
Example	`> mrsvctl -c -h node01 -T offset=200,etype=start,atype=exec,action=/tmp/email.sh` *Moab will create a reservation on `node01` and fire the script `/tmp/email.sh` `200` seconds after it starts.*

-V	
Name	VARIABLE
Format	`<name>`[=`<value>`][[;`<name>`[=`<value>`]]...]
Description	Semicolon-delimited list of variables that will be set when the reservation is created (see 17.4 Trigger Variables for more information). Names with no values will simply be set to `TRUE`.
Example	`> mrsvctl -c -h node01 -V $T1=mac;var2=18.19` *Moab will create a reservation on `node01` and set `$T1` to `mac` and `var2` to `18.19`.* For information on modifying a variable on a reservation, see MODIFY above.

-x	
Name	JOBLIST
Format	-x `<jobs to be excluded>`
Description	The `-x` flag, when used with `-F ignjobrsv`, lets users create reservations but exclude certain nodes that are running the listed jobs. The `-F` flag instructs *`mrsvctl`* to still consider nodes with current running jobs. The nodes are not listed directly.
Example	`> mrsvctl -c -t 5 -F ignjobrsv -x moab.5,moab.6` *Moab will create the reservation while assigning the nodes. Nodes running jobs `moab5` and `moab6` will not be assigned to the reservation.*

3.7.24.E Parameters

RESERVATION PATTERN	
Format	<STRING>

RESERVATION PATTERN	
Description	A pattern specifying the reservation(s) to be affected by this action consisting of a space-delimited list of one or more of the following reservation expressions: • The name of a reservation. • The string "ALL", which matches all reservations. • A regular expression matching zero or more reservations. A reservation expression is treated as a regular expression if it has a prefix of "x:" or if it contains one of the characters in "[ ] ( ) * ^ $,". Moab does a case-insensitive match using POSIX extended regular expressions and will match any part of the reservation name unless anchored with "^" or "$".
Example	`'^system'` *Specifies all reservations starting with the word "system".*

3.7.24.F Resource Allocation Details

When allocating resources, the following rules apply:

- When specifying tasks, each task defaults to one full compute node unless otherwise specified using the -R specification.
- When specifying tasks, the reservation will not be created unless all requested resources can be allocated (this behavior can be changed by specifying -F besteffort).
- When specifying tasks or hosts, only nodes in an idle or running state will be considered (this behavior can be changed by specifying -F ignstate).

3.7.24.G Reservation Timeframe Modification

Moab supports dynamically modifying the timeframe of existing reservations. This can be accomplished using the mrsvctl -m flag. By default, Moab will perform advanced boundary and resource access to verify that the modification does not result in an invalid scheduler state. However, in certain circumstances admins may want to FORCE the modification in spite of any access violations. This can be done using the switch `mrsvctl -m --flags=force`, which forces Moab to bypass any access verification and force the change through.

3.7.24.H Extending a Reservation by Modifying the Endtime

The following increases the endtime of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start          End    Duration     N/P     StartTime
system.1            User -    11:35:57  1:11:35:57  1:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime+=24:00:00 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start          End    Duration     N/P     StartTime
system.1            User -    11:35:23  2:11:35:23  2:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
```

The following increases the endtime of a reservation by setting the endtime to an absolute time:

```
$> showres
ReservationID       Type S       Start          End    Duration     N/P     StartTime
system.1            User -    11:33:18  1:11:33:18  1:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime=0_11/20 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start          End    Duration     N/P     StartTime
system.1            User -    11:33:05  2:11:33:05  2:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
```

3.7.24.I Extending a Reservation by Modifying the Duration

The following increases the duration of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start          End    Duration     N/P     StartTime
system.1            User -    11:28:46  1:11:28:46  1:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration+=24:00:00 system.1
duration for rsv 'system.1' changed
>$ showres
ReservationID       Type S       Start          End    Duration     N/P     StartTime
system.1            User -    11:28:42  2:11:28:42  2:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
```

The following increases the duration of a reservation by setting the duration to an absolute time:

```
$> showres
ReservationID       Type S       Start          End    Duration     N/P     StartTime
system.1            User -    11:26:41  1:11:26:41  1:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration=48:00:00 system.1
duration for rsv 'system.1' changed
$> showres
```

```
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:26:33  2:11:26:33  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

3.7.24.J Shortening a Reservation by Modifying the Endtime

The following modifies the endtime of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:15:51  2:11:15:51  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime-=24:00:00 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:15:48  1:11:15:48  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following modifies the endtime of a reservation by setting the endtime to an absolute time:

```
$ showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:14:00  2:11:14:00  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m endtime=0_11/19 system.1
endtime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:13:48  1:11:13:48  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

3.7.24.K Shortening a Reservation by Modifying the Duration

The following modifies the duration of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:12:20  2:11:12:20  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration-=24:00:00 system.1
duration for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:12:07  1:11:12:07  1:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
```

The following modifies the duration of a reservation by setting the duration to an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration     N/P     StartTime
system.1            User -    11:10:57  2:11:10:57  2:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m duration=24:00:00 system.1
duration for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration     N/P     StartTime
system.1            User -    11:10:50  1:11:10:50  1:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
```

3.7.24.L Modifying the Starttime of a Reservation

The following increases the starttime of a reservation using the += tag:

```
$> showres
ReservationID       Type S       Start         End    Duration     N/P     StartTime
system.1            User -    11:08:30  2:11:08:30  2:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime+=24:00:00 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration     N/P     StartTime
system.1            User -  1:11:08:23  3:11:08:23  2:00:00:00     1/2     Sun Nov 19
00:00:00
1 reservation located
```

The following decreases the starttime of a reservation using the -= tag:

```
$> showres
ReservationID       Type S       Start         End    Duration     N/P     StartTime
system.1            User -    11:07:04  2:11:07:04  2:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime-=24:00:00 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration     N/P     StartTime
system.1            User -   -12:53:04  1:11:06:56  2:00:00:00     1/2     Fri Nov 17
00:00:00
1 reservation located
```

The following modifies the starttime of a reservation using an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration     N/P     StartTime
system.1            User -    11:05:31  2:11:05:31  2:00:00:00     1/2     Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime=0_11/19 system.1
starttime for rsv 'system.1' changed
```

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -  1:11:05:18  3:11:05:18  2:00:00:00    1/2    Sun Nov 19
00:00:00
1 reservation located
```

The following modifies the starttime of a reservation using an absolute time:

```
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -    11:04:04  2:11:04:04  2:00:00:00    1/2    Sat Nov 18
00:00:00
1 reservation located
$> mrsvctl -m starttime=0_11/17 system.1
starttime for rsv 'system.1' changed
$> showres
ReservationID       Type S       Start         End    Duration    N/P    StartTime
system.1            User -   -12:56:02  1:11:03:58  2:00:00:00    1/2    Fri Nov 17
00:00:00
1 reservation located
```

3.7.24.M Examples

- Basic Reservation
- System Maintenance Reservation
- Explicit Task Description
- Dynamic Reservation Modification
- Reservation Modification
- Allocating Reserved Resources
- Modifying an Existing Reservation

Example 3-19: Basic Reservation

Reserve two nodes for use by users `john` and `mary` for a period of 8 hours starting in 24 hours:

```
> mrsvctl -c -a USER=john,USER=mary -starttime +24:00:00 -duration 8:00:00 -t 2
reservation 'system.1' created
```

Example 3-20: System Maintenance Reservation

Schedule a system wide reservation to allow a system maintenance on Jun 23, 8:00 A.M. until Jun 23, 5:00 P.M.:

```
% mrsvctl -c -s 8:00:00_06/23 -e 17:00:00_06/23 -h ALL
reservation 'system.1' created
```

Example 3-21: Explicit Task Description

Reserve 1 processor and `512` MB of memory on nodes `node003` through node `node006` for members of the group `staff` and jobs in the `interactive` class:

```
> mrsvctl -c -R PROCS=1,MEM=512 -a GROUP=staff,CLASS=interactive -h 'node00[3-6]'
reservation 'system.1' created
```

Example 3-22: Dynamic Reservation Modification

Modify reservation `john.1` to start in 2 hours, run for 2 hours, and include `node02` in the hostlist:

```
> mrsvctl -m starttime=+2:00:00,duration=2:00:00,HostExp+=node02
Note:  hosts added to rsv system.3
```

Example 3-23: Reservation Modification

Remove user `John`'s access to reservation `system.1`:

```
> mrsvctl -m -a USER=John system.1 --flags=unset
successfully changed ACL for rsv system.1
```

Example 3-24: Allocating Reserved Resources

Allocate resources for group `dev` that are exclusive except for resources found within reservations `myrinet.3` or `john.6`:

```
> mrsvctl -c -E -a group=dev,rsv=myrinet.3,rsv=john.6 -h 'node00[3-6]'
reservation 'dev.14' created
```

Create exclusive `network` reservation on racks 3 and 4:

```
> mrsvctl -c -E -a group=ops -g network -f rack3 -h ALL
reservation 'ops.1' created
> mrsvctl -c -E -a group=ops -g network -f rack4 -h ALL
reservation 'ops.2' created
```

Allocate `64` nodes for 2 hours to new reservation and grant access to reservation `system.3` and all reservations in the reservation group `network`:

```
> mrsvctl -c -E -d 2:00:00 -a group=dev -t 64 -S rsvaccesslist=system.3,network
reservation 'system.23' created
```

Allocate `4` nodes for 1 hour to new reservation and grant access to idle job reservations:

```
> mrsvctl -c -E -d 1:00:00 -t 4 -S rsvaccesslist=idle
reservation 'system.24' created
```

Example 3-25: Modifying an Existing Reservation

Remove user `john` from reservation ACL:

```
> mrsvctl -m -a USER=john system.1 --flags=unset
```

```
successfully changed ACL for rsv system.1
```

Change reservation group:

```
> mrsvctl -m RSVGROUP=network ops.4
successfully changed RSVGROUP for rsv ops.4
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 6.1.2 Administrative Reservations
- 3.7.33 showres
- 3.7.13 mdiag -r
- 3.7.27 mshow -a command to identify available resources

3.7.25 mschedctl

3.7.25.A Synopsis

mschedctl -A '<MESSAGE>'

mschedctl -c message `messagestring` [-o `type`:`val`]

mschedctl -c trigger `triggerid` -o `type`:`val`

mschedctl -d trigger:`triggerid`

mschedctl -d message:`index`:wq

mschedctl -f {all|fairshare|usage}

mschedctl -k

mschedctl -l {config|feature|gmetric|gres|message|opsys|trigger|trans} [-v] [--xml]

mschedctl -L [`<LOGLEVEL>`[:`<LOG_FILE>`]]

mschedctl -m config `string` [-e]

mschedctl -m trigger `triggerid` `attr=val`[,`attr=val`...]

mschedctl -q mschedctl -q pactions --xml

mschedctl -p

mschedctl -r [`resumetime`]

mschedctl -R

mschedctl -s [STOPITERATION]

mschedctl -S [STEPITERATION]

mschedctl -W

```
[--about] [--help] [--host=<serverHostName>] [--
loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>]
[--timeout=<seconds>] [--version] [--xml]
```

3.7.25.B Overview

The *mschedctl* command controls various aspects of scheduling behavior. It is used to manage scheduling activity, shutdown the scheduler, and create resource trace files. It can also evaluate, modify, and create parameters, triggers, and messages.

With many flags, the `--msg=<MSG>` option can be specified to annotate the action in the event log.

3.7.25.C Options

-A - ANNOTATE	
Format	<STRING>
Description	Report the specified parameter modification to the event log and annotate it with the specified message. The RECORDEVENTLIST parameter must be set in order for this to work.
Example	`mschedctl -A 'increase logging' -m 'LOGLEVEL 6'` *Adjust the LOGLEVEL parameter and record an associated message.*

-c - CREATE	
Format	One of: • **message** <STRING> [-o <TYPE>:<VAL>] • **trigger**<TRIGSPEC> -o <OBJECTTYPE>:<OBJECTID> • gevent -n <NAME> [-m <message>]

-c - CREATE	
Description	Create a message, trigger, or gevent and attach it to the specified object. To create a trigger on a default object, use the Moab configuration file (`moab.cfg`) rather than the *`mschedctl`* command.

Example	`mschedctl -c message tell the admin to be nice`
	Create a message on the system table.
	
	`mschedctl -c trigger EType=start,AType=exec,Action="/tmp/email $OWNER $TIME" -o rsv:system.1`
	Create a trigger linked to `system.1`.
	
	Creating triggers on default objects via `mschedctl -c trigger` does not propagate the triggers to individual objects. To propagate triggers to all objects, the triggers must be created within the `moab.cfg` file; for example: `NODECFG[DEFAULT]TRIGGER`.
	
	`mschedctl -c gevent -n diskfailure -m "node=n4"`
	Create a gevent indicating a disk failure on the node labeled `n4`.

-d - DESTROY	
Format	One of: • **trigger**:`<TRIGID>` • **message**:`<INDEX>`
Description	Delete a trigger or message.

Example	`mschedctl -d trigger:3`
	Delete trigger 3.
	
	`mschedctl -d message:5`
	Delete message with index 5.

-f - FLUSH	
Format	{all|fairshare|usage}
Description	Reset all internally-stored Moab Scheduler statistics to the initial start-up state as of the time the command was executed. Flushing should only be used if you experience corrupt statistics. The best practice is to pause the Moab scheduler with *`mschedctl -p`* before running the flush command. After running the flush command, unpause the Moab scheduler with *`mschedctl -r`* and the jobs will start flowing again. For all external observers this will be a transparent flush unless they are watching the stats.
Example	`mschedctl -f usage` *Flush usage statistics.*

-k - KILL	
Description	Stop scheduling and exit the scheduler.
Example	`mschedctl -k` *Kill the scheduler.*

-l - LIST	
Format	{config|feature|gmetric|gres|message|opsys|trans|trigger} [-v] [--xml] Using the `--xml` argument with the trans option returns XML that states if the queried TID is valid or not.
Default	config
Description	List the generic metrics, generic resources, scheduler configuration, system messages, operating systems, triggers, transactions, or node features recognized by Moab. This command does not show credential parameters (such as user, group class, QoS, account).

-l - LIST	

Example	`mschedctl -l config`
	List system parameters.
	The `config` command without the `-v` flag does not show the settings of all scheduling parameters. To show the settings of all scheduling parameters, use the `-v` flag. This will provide an extended output. This output is often best used in conjunction with the `grep` command as the output can be voluminous.
	`mschedctl -l feature`
	List all node features recognized by Moab.
	`mschedctl -l gmetric`
	List all configured generic metrics.
	`mschedctl -l gres`
	List all configured generic resources. Use the -v flag to display generic resource traits (such as license or numa).
	`mschedctl -l message`
	List all system messages.
	`mschedctl -l opsys`
	List all recognized operating systems.
	`mschedctl -l trans 1`
	List transaction id `1`.
	`mschedctl -l trigger`
	List triggers.

-L - LOG	
Format	[`<LOGLEVEL>`[: `<LOG_FILE>`]]
Default	7 $MOABHOMEDIR/log/moab.log
Description	Create a temporary log file with the specified loglevel. If no log file is given, Moab creates a log file in the log directory whose filename extension is the

-L - LOG	
	timestamp of when the command was run (for example, /opt/moab/log/moab.log.20240405081227).
Example	`mschedctl -L7:/tmp/moab.log`

-m - MODIFY	
Format	One of: • **config** [`<STRING>`] [`-e`] `<STRING>` is any string that is acceptable in `moab.cfg` ◦ If no string is specified, `<STRING>` is read from STDIN. ◦ If `-e` is specified, the configuration string will be evaluated for correctness but no configuration changes will take place. Any issues with the provided string will be reported to STDERR. Use of mschedctl --flags=persistent -m <config> has been deprecated; use the following method instead: ◦ Run mschedctl -m <config> to put the change into effect dynamically. ◦ Manually add the settings to the moab.cfg file, so that it always goes into effect after any future Moab restarts/recycles. Dynamically modifying classes is not recommended. Moab should be restarted whenever classes are modified. This is especially true given the fact that sometimes the classes/queues/partitions are under control of a resource manager. For dynamic operations, use node sets/features or reservations. • **trigger**:`<TRIGID> <ATTR>=<VAL>` Where `<ATTR>` is one of action, atype, etype, iscomplete, oid, otype, offset, or threshold.
Description	Modify a system parameter or trigger.

-m - MODIFY	
	Moab only loads the following list of parameters when first starting up. Therefore, to change any of these, you must edit the setting in moab.cfg and then restart/recycle with mschedctl -R. • JOBMAXNODECOUNT • MAXGMETRIC • MAXGRES • MAXJOB • MAXNODE • MAXRSVPERNODE • STATPROC* • STATTIME*

Example	`mschedctl -m config LOGLEVEL 9`
	Change the system loglevel to 9.
	`mschedctl -m trigger:2 AType=exec,Offset=200,OID=system.1`
	Change aspects of trigger 2.

-p - PAUSE	
Description	Disable scheduling but allow the scheduler to update its cluster and workload state information.
Example	`mschedctl -p`

-q QUERY PENDING ACTIONS	
Default	`mschedctl -q pactions --xml`
Description	A way to view pending actions. Only an XML request is valid. Pending actions can be system jobs.
Example	`mschedctl -q pactions --xml`

-r - RESUME	
Format	`mschedctl -r` [[HH:[MM:]]SS]
Default	0
Description	Resume scheduling in the specified amount of time (or immediately if none is specified).
Example	`mschedctl -r` *Resume scheduling immediately.*

-R - RECYCLE	
Description	Recycle scheduler immediately (shut it down and restart it using the original execution environment and command line arguments). If Moab has been started under systemd, use `systemctl restart moab.service` instead of using this option.
Example	`mschedctl -R` *Recycle scheduler immediately.* To restart Moab with its last known scheduler state, use: `mschedctl -R savestate`

-s - STOP	
Format	`<INTEGER>`
Default	0
Description	Suspend/stop scheduling at specified iteration (or at the end of the current iteration if none is specified). If the letter I follows `<ITERATION>`, Moab will not process client requests until this iteration is reached.
Example	`mschedctl -s 100I` *Stop scheduling at iteration 100 and ignore all client requests until*

-s - STOP	
	then.

-S - STEP	
Format	`<INTEGER>`
Default	0
Description	Step the specified number of iterations (or to the next iteration if none is specified) and suspend scheduling. If the letter I follows `<ITERATION>`, Moab will not process client requests until this iteration is reached.
Example	`mschedctl -S` *Step to the next iteration and stop scheduling.*

-W	
Description	Preform a manual checkpoint file write.
Example	`mschedctl -W`

3.7.25.D Example

Example 3-26: Shutting down the Scheduler

```
mschedctl -k
scheduler will be shutdown immediately
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes

3.7.26 mshow

3.7.26.A Synopsis

mshow [-a] [-q jobqueue=active] [--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]

3.7.26.B Overview

The *mshow* command displays various diagnostic messages about the system and job queues.

3.7.26.C Arguments

Flag	Description
-a	AVAILABLE RESOURCES
-q [<QUEUENAME>]	Displays the job queues.

3.7.26.D Format

AVAILABLE RESOURCES	
Format	Can be combined with `--flags=[tid|verbose|future] --format=xml` and/or `-w`
Description	Display available resources.
Example	`> mshow -a -w user=john --flags=tid --format=xml` *Show resources available to john in XML format with a transaction id. See 3.7.27 mshow -a for details.*

JOB QUEUE	
Format	<QUEUENAME>, where the queue name is one of: active, eligible, or blocked. Job queue names can be delimited by a comma to display multiple queues. If no job queue name is specified, mshow displays all job queues.
Description	Displays the job queues. If a job queue name is specified, *mshow* shows only that job queue.

Example	`> mshow -q active,blocked`
	 `[Displays all jobs in the active and blocked queues]`
	 `...`

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.27 mshow -a - command that shows available resources

3.7.27 mshow -a

3.7.27.A Synopsis

mshow -a [-i] [-o] [-T] [-w where] [-x] [--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]

3.7.27.B Overview

The *mshow -a* command allows for querying of available system resources.

3.7.27.C Arguments

[-i]	INTERSECTION
[-o]	NO AGGREGATE
[-T]	TIMELOCK

[-w]	WHERE
[-x]	EXCLUSIVE

Table 3-1: Argument Format

--flags	
Name	Flags
Format	--flags=[future | policy | tid | summary | verbose]
Description	**future** will return resources available immediately and available in the future. **policy (Deprecated. May be removed in a future release.)** will apply charging policies to determine the total cost of each reported solution (only enabled for XML responses). **tid** will associate a transaction ID with the reported results. **summary** will assign all jointly allocated transactions as dependencies of the first transaction reported. **verbose** will return diagnostic information.
Example	`> mshow -a -w user=john --flags=tid --xml` *Show resources available to john in XML format with a transaction ID.*

--xml	
Name	XML
Format	--xml
Description	Report results in XML format.
Example	`> mshow -a -w user=john --flags=tid --xml` *Show resources available to* `john` *in XML format with a transaction ID.*

-i	
Name	INTERSECTION
Description	Specifies that an intersection should be performed during an *mshow -a* command with multiple requirements.

-o	
Name	NO AGGREGATE
Description	Specifies that the results of the command *mshow -a* with multiple requirements should not be aggregated together.

-T	
Name	TIMELOCK
Description	Specifies that the multiple requirements of an `mshow -a` command should be timelocked.

Example	`> mshow -a -w minprocs=1,os=linux,duration=1:00:00 \`
	`  -w minprocs=1,os=aix,duration=10:00 \`
	`  --flags=tid,future -x -T`

-w	
Name	WHERE
Format	Comma-delimited list of `<ATTR>=<VAL>` pairs: `<ATTR>=<VAL>` [,`<ATTR>=<VAL>`]... If any of the `<ATTR>=<VAL>` pairs contains a sub-list that is also comma-delimited, the entire `-w` string must be wrapped in single quotations with the sub-list expression wrapped in double quotations. See the example below. Attributes are listed below in Table 3-2: Request Attributes.
Description	Add a Where clause to the current command (currently supports up to six co-allocation clauses).
Example	`> mshow -a -w minprocs=2,duration=1:00:00 -w nodemem=512,duration=1:00:00`

-w	
	Moab returns a list of all nodes with at least 2 processors and one hour duration or with a memory of 512 and a duration of one hour. `> mshow -a -w nodefeature=\!vmware:gpfs --flags=future` *Moab returns a list of all nodes that do not contain the `vmware` feature but that do contain the `gpfs` feature.* `> mshow -a -w 'duration=INFINITY,"excludehostlist=n01,n12,n23"'` *Moab returns a list of all nodes with a duration of `INFINITY`, except for nodes named n01, n12, and n23.* *Note the use of single quotations containing the entire `-w` string and the use of double quotations containing the `excludehostlist` attribute.*

-x	
Name	EXCLUSIVE
Description	Specifies that the multiple requirements of an *`mshow -a`* command should be exclusive (i.e., each node can only be allocated to a single requirement).
Example	`> mshow -a -w minprocs=1,os=linux -w minprocs=1,os=aix --flags=tid -x`

Table 3-2: Request Attributes

Name	Description
account	The account credential of the requestor.
acl	ACL to attach to the reservation. This ACL must be enclosed in quotation marks. For example: `$ mshow -a ... -w acl=\"user=john\" ...`
arch	Select only nodes with the specified architecture.
class	The class credential of the requestor.
coalloc	The co-allocation group of the specific Where request (can be any string but must match co-allocation group of at least one other Where request).

Name	Description
	The number of tasks requested in each Where request must be equal whether this taskcount is specified via `minprocs`, `mintasks`, or `gres`.
count	The number of profiles to apply to the resource request.
displaymode	Possible value is `future`. (Example: `displaymode=future`). Constrains how results are presented; setting `future` evaluates which resources are available now and which resources will be available in the future that match the requested attributes.
duration	The duration for which the resources will be required in format `[[[DD:]HH:]MM:]SS`
excludehostlist	Do not select any nodes from the given list. The list must be comma-delimited. `> mshow -a -w 'duration=INFINITY,"excludehostlist=n01,n12,n23"'` *Moab returns a list of all nodes with a duration of `INFINITY`, except for nodes named n01, n12, and n23.* *Note the use of single quotations to contain the entire `-w` string, and the use of double quotations containing the `excludehostlist` attribute.*
gres	Select only nodes that possess the specified generic resource.
group	The group credential of the requestor.
hostlist	Select only the specified resources. The list must be comma-delimited. `> mshow -a -w 'duration=INFINITY,"hostlist=n01,n12,n23"'` *Moab returns a list of nodes from the selected hostlist that have a duration of `INFINITY`.* *Note the use of single quotations to contain the entire `-w` string, and the use of double quotations containing the `hostlist` attribute.*
job	Use the resource, duration, and credential information for the job specified as a resource request template.
jobfeature	Select only resources that allow access to jobs with the specified job

Name	Description
	features.
jobflags	Select only resources that allow access to jobs with the specified job flags. The `jobflags` attribute accepts a colon-delimited list of multiple flags.
minnodes	Return only results with at least the number of nodes specified. If used with TIDs, return only solutions with exactly minnodes nodes available.
minprocs	Return only results with at least the number of processors specified. If used with TIDs, return only solutions with exactly minprocs processors available.
mintasks	**FORMAT**: `<TASKCOUNT>`[@`<RESTYPE>`:`<COUNT>`[+`<RESTYPE>`:`<COUNT>`]...], where `<RESTYPE>` is one of `procs`, `mem`, `disk`, or `swap`. Return only results with at least the number of tasks specified. If used with TIDs, return only solutions with exactly mintasks available.
nodedisk	Select only nodes with at least nodedisk MB of local disk configured.
nodefeature	Select only nodes with all specified features present and nodes without all `\!` specified features using format `[\!]<feature>[:[\!]<feature>]...` You must set the future flag when specifying node features.
nodemem	Select only nodes with at least `nodemem` MB of memory configured.
offset	Select only resources that can be co-allocated with the specified time offset where offset is specified in the format `[[[DD:]HH:]MM:]SS`
os	Select only nodes with have, or can be provisioned to have, the specified operating system.
partition	The partition where the resources must be located.
policylevel	Enable policy enforcement at the specified policy constraint level.
qos	The qos credential of the requestor.
rsvprofile	Use the specified profile if committing a resulting transaction ID directly to a reservation.
starttime	Constrain the timeframe for the returned results by specifying one or more

Name	Description
	ranges using the format `<STIME>`[-`<ENDTIME>`][;`<STIME>`[-`<ENDTIME>`]] where each time is specified in the format in absolute, relative, or epoch time format (`[HH[:MM[:SS]]][_MO[/DD[/YY]]]` or `+[[[DD:]HH:]MM:]SS` or `<EPOCHTIME>`). The starttime specified is not the exact time at which the returned range must start, but is rather the earliest possible time the range can start.
taskmem	Require `taskmem` MB of memory per task located.
tpn	Require exactly `tpn` tasks per node on all discovered resources.
user	The user credential of the requestor.
var	Use associated variables in generating per transaction charging quotes.
variables	Takes a string of the format `variables='var[=attr]'[;'var[=attr]'` and passes the variables onto the reservation when used in conjunction with `--flags=tid` and `mrsvctl -c -R <tid>`.

3.7.27.D Usage Notes

The *`mshow -a`* command allows for querying of available system resources. When combined with the `--flags=tid` option these available resources can then be placed into a packaged reservation (using mrsvctl -c -R). This allows system admins to grab and reserve available resources for whatever reason, without conflicting with jobs or reservations that may be holding certain resources.

There are a few restrictions on which `<ATTR>` from the -w command can be placed in the same req: `minprocs`, `minnodes`, and `gres` are all mutually exclusive, only one can be used per `-w` request.

The allocation of available nodes will follow the global parameter NODEALLOCATIONPOLICY.

When the '-o' flag is not used, multi-request results will be aggregated. This aggregation will negate the use of offsets and request-specific starttimes.

The config parameter RESOURCEQUERYDEPTH controls the maximum number of options that will be returned in response to a resource query.

3.7.27.E Examples

Example 3-27: Basic Compute Node Query and Reservation

```
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobfeature=shared --
flags=tid,future

Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:28:09_04/27  TID=4  ReqID=0
ALL               1      1      10:00:00      10:00:00  17:14:48_04/28  TID=5  ReqID=0
ALL               1      1      10:00:00      20:00:00  21:01:27_04/29  TID=6  ReqID=0
> mrsvctl -c -R 4
Note:  reservation system.2 created
```

Example 3-28: Mixed Processor and License Query

Select one node with 4 processors and 1 matlab license where the matlab license is only available for the last hour of the reservation. Also, select 16 additional processors that are available during the same timeframe but which can be located anywhere in the cluster. Group the resulting transactions together using transaction dependencies so only the first transaction needs to be committed to reserve all associated resources.

```
> mshow -a -i -o -x -w mintasks=1@PROCS:4,duration=10:00:00,coalloc=a \
                    -w gres=matlab,offset=9:00:00,duration=1:00:00,coalloc=a \
                    -w minprocs=16,duration=10:00:00 --flags=tid,future,summary
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:28:09_04/27  TID=4  ReqID=0
ALL               1      1      10:00:00      10:00:00  17:14:48_04/28  TID=5  ReqID=0
ALL               1      1      10:00:00      20:00:00  21:01:27_04/29  TID=6  ReqID=0
> mrsvctl -c -R 4

Note:  reservation system.2 created
Note:  reservation system.3 created
Note:  reservation system.4 created
```

Example 3-29: Request for Generic Resources

Query for a generic resource on a specific host (no processors, only a generic resource):

```
> mshow -a -i -x -o -w gres=dvd,duration=10:00,hostlist=node03 --flags=tid,future
Partition     Tasks  Nodes   StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      00:00:00      00:10:00  11:33:25_07/27  TID=16
ReqID=0
ALL               1      1      00:10:00      00:10:00  11:43:25_07/27  TID=17
ReqID=0
ALL               1      1      00:20:00      00:10:00  11:53:25_07/27  TID=18
ReqID=0
> mrsvctl -c -R 16
Note:  reservation system.6 created
> mdiag -r system.6
Diagnosing Reservations
RsvID                      Type Par   StartTime     EndTime     Duration Node Task
Proc
-----                      ---- ---   ---------     -------     -------- ---- ---- ---
```

```
-
system.6                   User loc   -00:01:02    00:08:35     00:09:37    1    1
0
    Flags: ISCLOSED
    ACL:   RSV==system.6=
    CL:    RSV==system.6
    Accounting Creds:  User:test
    Task Resources: dvd: 1
    Attributes (HostExp='^node03$')
    Rsv-Group: system.6
```

Example 3-30: Allocation of Shared Resources

This example walks through a relatively complicated example where a set of resources can be reserved to be allocated for shared requests. In the example below, the first *mshow* query looks for resources within an existing shared reservation. In the example, this first query fails because there is now existing reservation. The second query looks for resources within an existing shared reservation. In the example, this first query fails because there is now existing reservation. The second *mshow* request asks for resources outside of a shared reservation and finds the desired resources. These resources are then reserved as a shared pool. The third *mshow* request again asks for resources inside of a shared reservation and this time finds the desired resources.

```
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobflags=ADVRES,jobfeature=shared
--flags=tid

Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
> mshow -a -w duration=100:00:00,minprocs=1,os=AIX53,jobfeature=shared --flags=tid
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1     100:00:00      00:00:00  13:20:23_04/27   TID=1   ReqID=0
> mrsvctl -c -R 1
Note:  reservation system.1 created
> mshow -a -w duration=10:00:00,minprocs=1,os=AIX53,jobflags=ADVRES,jobfeature=shared
--flags=tid
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               1      1      10:00:00      00:00:00  13:20:36_04/27   TID=2   ReqID=0
> mrsvctl -c -R 2
Note:  reservation system.2 created
```

Example 3-31: Full Resource Query in XML Format

The following command will report information on all available resources that meet at least the minimum specified processor and walltime constraints and which are available to the specified user. The results will be reported in XML to allow for easy system processing.

```
> mshow -a -w class=grid,minprocs=8,duration=20:00 --format=xml --flags=future,verbose

<Data>
  <Object>cluster</Object>
  <job User="john" time="1162407604"></job>
  <par Name="template">
    <range duration="Duration" nodecount="Nodes" proccount="Procs"
starttime="StartTime"></range>
```

```
    </par>
  <par Name="ALL" feasibleNodeCount="131" feasibleTaskCount="163">
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
041:1,opt-042:1,x86-001:1,P690-001:1,P690-021:1,P690-022:1"
        index="0" nodecount="10" proccount="8" reqid="0"
starttime="1162407604"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,x86-001:1,P690-001:1,P690-021:1,P690-022:1"
        index="0" nodecount="11" proccount="8"reqid="0"
starttime="1162411204"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,x86-001:1,x86-002:1,x86-004:1,
        x86-006:1,x86-013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1,P690-001:1,P690-
021:1,P690-022:1"
        index="0" nodecount="19" proccount="8" reqid="0"
starttime="1162425519"></range>
    </par>
  <par Name="SharedMem">
    <range duration="1200" hostlist="P690-001:1,P690-002:1,P690-003:1,P690-004:1,P690-
005:1,P690-006:1,P690-007:1,P690-008:1,P690-009:1,
        P690-010:1,P690-011:1,P690-012:1,P690-013:1,P690-014:1,P690-015:1,P690-
016:1,P690-017:1,P690-018:1,P690-019:1,P690-020:1,P690-021:1,
        P690-022:1,P690-023:1,P690-024:1,P690-025:1,P690-026:1,P690-027:1,P690-
028:1,P690-029:1,P690-030:1,P690-031:1,P690-032:1"
        index="0" nodecount="32" proccount="8" reqid="0"
starttime="1163122507"></range>
    </par>
  <par Name="64Bit">
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1"
        index="0" nodecount="7" proccount="8" reqid="0"
starttime="1162411204"></range>
    <range duration="1200" hostlist="opt-001:1,opt-024:1,opt-025:1,opt-027:2,opt-
039:1,opt-041:1,opt-042:1,opt-043:1,opt-044:1,opt-045:1,
        opt-046:1,opt-047:1,opt-048:1,opt-049:1,opt-050:1"
        index="0" nodecount="15" proccount="8" reqid="0"
starttime="1162428996"></range>
    <range duration="1200" hostlist="opt-001:1,opt-006:1,opt-007:2,opt-008:2,opt-
009:2,opt-010:2,opt-011:2,opt-012:2,opt-013:2,opt-014:2,
        opt-015:2,opt-016:2,opt-017:2,opt-018:2,opt-019:2,opt-020:2,opt-021:2,opt-
022:2,opt-023:2,opt-024:2,opt-025:1,opt-027:2,opt-039:1,
        opt-041:1,opt-042:1,opt-043:1,opt-044:1,opt-045:1,opt-046:1,opt-047:1,opt-
048:1,opt-049:1,opt-050:1"
        index="0" nodecount="33" proccount="8" reqid="0"
starttime="1162876617"></range>
    </par>
  <par Name="32Bit">
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1"
        index="0" nodecount="9" proccount="8" reqid="0"
starttime="1162425519"></range>
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-037:1,x86-042:1,x86-043:1"
        index="0" nodecount="11" proccount="8" reqid="0"
starttime="1162956803"></range>
    <range duration="1200" hostlist="x86-001:1,x86-002:1,x86-004:1,x86-006:1,x86-
013:1,x86-014:1,x86-015:1,x86-016:1,x86-027:1,x86-028:1,
        x86-029:1,x86-030:1,x86-037:1,x86-041:1,x86-042:1,x86-043:1,x86-046:1,x86-
047:1,x86-048:1,x86-049:1"
        index="0" nodecount="20" proccount="8" reqid="0"
starttime="1163053393"></range>
    </par>
  </Data>
```

This command reports the original query, and the timeframe, resource size, and hostlist associated with each possible time slot.

3.7.27.F Basic Current and Future Requests

The *mshow* command can report information on many aspects of the scheduling environment. To request information on available resources, the `-a` flag should be used. By default, the *mshow* command resource availability query only reports resources that are immediately available. To request information on specific resources, the type of resources required can be specified using the `-w` flag as in the following example:

```
> mshow -a -w taskmem=1500,duration=600
...
```

To view current and future resource availability, the `future` flag should be set as in the following example:

```
> mshow -a -w taskmem=1500,duration=600 --flags=future
...
```

Co-Allocation Resources Queries

In many cases, a particular request will need simultaneous access to resources of different types. The *mshow* command supports a co-allocation request specified by using multiple `-w` arguments. For example, to request 16 nodes with feature `fastcpu` and 2 nodes with feature `fastio`, the following request might be used:

```
> mshow -a -w minprocs=16,duration=1:00:00,nodefeature=fastcpu -w
minprocs=2,nodefeature=fastio,duration=1:00:00 --flags=future
Partition      Procs  Nodes   StartOffset      Duration       StartDate
---------      -----  -----  ------------  ------------  --------------
ALL               16      8      00:00:00       1:00:00  13:00:18_08/25  ReqID=0
ALL                2      1      00:00:00       1:00:00  13:00:18_08/25  ReqID=1
```

The mshow -a topic contains a list of the different resources that can be queried, and examples on using *mshow*.

Using Transaction IDs

By default, the *mshow* command reports simply when and where the requested resources are available. However, when the `tid` flag is specified, the *mshow* command returns both resource availability information and a handle to these resources called a Transaction ID as in the following example:

```
> mshow -a -w minprocs=16,nodefeature=fastcpu,duration=2:00:00 --flags=future,tid
```

```
Partition      Procs  Nodes   StartOffset      Duration       StartDate
---------      -----  -----  -------------  -------------  ---------------
ALL               16     16       00:00:00        2:00:00  13:00:18_08/25  TID=26 ReqID=0
```

In the preceding example, the returned transaction ID (TID) can then be used to reserve the available resources using the mrsvctl -c -R command:

```
> mrsvctl -c -R 26
reservation system.1 successfully created
```

Any TID can be printed out using the mschedctl -l trans command:

```
Code example (replace with your own content)
> mschedctl -l trans 26
TID[26]  A1='node01'  A2='600'  A3='1093465728'  A4='ADVRES'  A5='fastio'
```

Where `A1` is the hostlist, `A2` is the duration, `A3` is the starttime, `A4` are any flags, and `A5` are any features.

Using Reservation Profiles

Reservation profiles (RSVPROFILE) stand as templates against which reservations can be created. They can contain a hostlist, startime, endtime, duration, access-control list, flags, triggers, variables, and most other attributes of an Administrative Reservation. The following example illustrates how to create a reservation with the exact same trigger-set:

```
-----
# moab.cfg
-----
RSVPROFILE[test1] TRIGGER=Sets=$Var1.$Var2.$Var3.!Net,EType=start,AType=exec,
        Action=/tmp/host/triggers/Net.sh,
        Timeout=1:00:00
RSVPROFILE[test1]                                  TRIGGER=Requires=$Var1.$Var2.$Var3,
        Sets=$Var4.$Var5,EType=start,
        AType=exec,Action=/tmp/host/triggers/
        FS.sh+$Var1:$Var2:$Var3,Timeout=20:00
RSVPROFILE[test1]
TRIGGER=Requires=$Var1.$Var2.$Var3.$Var4.$Var5,
        Sets=!NOOSinit.OSinit,Etype=start,
        AType=exec,
        Action=/tmp/host/triggers/
        OS.sh+$Var1:$Var2:$Var3:$Var4:$Var5
RSVPROFILE[test1]
TRIGGER=Requires=NOOSini,AType=cancel,EType=start
RSVPROFILE[test1]
TRIGGER=EType=start,Requires=OSinit,AType=exec,
        Action=/tmp/host/triggers/success.sh
...
-----
```

To create a reservation with this profile, the mrsvctl -c -P command is used:

```
> mrsvctl -c -P test1

reservation system.1 successfully created
```

Using Reservation Groups

Reservation groups are a way for Moab to tie reservations together. When a reservation is created using multiple Transaction IDs, these transactions and their resulting reservations are tied together into one group.

```
> mrsvctl -c -R 34,35,36
reservation system.99 successfully created
reservation system.100 successfully created
reservation system.101 successfully created
```

In the preceding example, these three reservations will be tied together into a single group. The mdiag -r command can be used to see which group a reservation belongs to. The mrsvctl -q diag -g command can also be used to print out a specific group of reservations. The mrsvctl -c -g command can also be used to release a group of reservations.

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.26 mshow

3.7.28 msub

3.7.28.A Synopsis

`msub` [-a *datetime*] [-A *account*] [-b *retry_count*] [-c *interval*] [-C *directive_prefix*] [-d *path*] [-e *path*] [-E] [-F] [-h] [-I] [-j *join*] [-k *keep*] [-K] [-l *resourcelist*] [-L *NUMA_resourcelist*] [-m *mailoptions*] [-M *user_list*] [-n *node_exclusive*] [-N *name*] [-o *path*] [-p *priority*] [-P `<user>`] [-q *destination*] [-r *yn*] [-S *pathlist*] [-t *jobarrays*] [-u *userlist*] [-v *variablelist*] [-V] [-w *<path>*] [-W *additionalattributes*] [-x] [-z] [--stagein] [--stageout] [--stageinfile] [--stageoutfile] [--stageinsize] [--stageoutsize] [--workflowjobids] [*script*] `[--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]`

3.7.28.B Overview

`msub` allows users to submit jobs directly to Moab. When a job is submitted directly to a resource manager (such as Torque), it is constrained to run on only those nodes that the resource manager is directly monitoring. In many instances, you may be controlling multiple resource managers. When a job is submitted to Moab rather than to a specific resource manager, it is not constrained as to what nodes it is executed on. `msub` can accept command line arguments (with the same syntax as qsub), job scripts (in either PBS or LoadLeveler syntax), or the SSS Job XML specification.

Moab must run as a root user in order for `msub` submissions to work. Workload submitted via msub when Moab is running as a non-root user fail immediately.

Submitted jobs can then be viewed and controlled via the mjobctl command.

Flags specified in the following table are not necessarily supported by all resource managers.

3.7.28.C Access

When Moab is configured to run as root, any user can submit jobs via `msub`.

3.7.28.D Options

-a	
Name	Eligible Date
Format	`[[[[CC]YY]MM]DD]hhmm[.SS]`
Description	Declares the time after which the job is eligible for execution.
Example	`> msub -a 12041300 cmd.pbs` *Moab will not schedule the job until 1:00 pm on December 4, of the current year.*

-A	
Name	Account
Format	`<ACCOUNT NAME>`
Description	Defines the account associated with the job.
Example	`> msub -A research cmd.pbs` *Moab will associate this job with account research.*

-b	
Name	Retry count
Format	`<retry_count>`
Description	Defines the number of times msub should retry connecting to the server.
Example	`> msub -b 5 cmd.pbs` *Moab will attempt to retry connecting to the server 5 times.*

-c	
Name	Checkpoint Interval
Format	`[n|s|c|c=<minutes>]`
Description	Checkpoint of the job will occur at the specified interval. **n** — No Checkpoint is to be performed. **s** — Checkpointing is to be performed only when the server executing the job is shut down. **c** — Checkpoint is to be performed at the default minimum time for the server executing the job. **c=<minutes>** — Checkpoint is to be performed at an interval of minutes.
Example	`> msub -c c=12 cmd.pbs` *The job will be checkpointed every 12 minutes.*

-C	
Name	Directive Prefix
Format	`'<PREFIX NAME>'`
Default	First known prefix (#PBS, #@, #BSUB, #!, #MOAB, #MSUB).
Description	Specifies which directive prefix should be used from a job script: • It is best to submit with single quotes. '#PBS' • An empty prefix will cause Moab to not search for any prefix. -C '' • Command line arguments have precedence over script arguments.

-C	
	• Custom prefixes can be used with the -C flag. -C '#MYPREFIX' • Custom directive prefixes must use PBS syntax. • If the -C flag is not given, Moab will take the first default prefix found. Once a directive is found, others are ignored.

Example	`> msub -C '#MYPREFIX' cmd.pbs`
	`#MYPREFIX -l walltime=5:00:00 (in cmd.pbs)`
	Moab will use the #MYPREFIX directive specified in cmd.pbs, setting the wallclock limit to five hours.

-d	
Name	Initial Working Directory
Format	`<path>`
Default	Depends on the resource manager being used. If using Torque, the default is $HOME.
Description	Specifies which directory the job should execute in.
Example	`> msub -d /home/test/job12 cmd.pbs` *The job will begin execution in the `/home/test/job12` directory.*

-e	
Name	Error Path
Format	`[<hostname>:]<path>`
Default	`$SUBMISSIONDIR/$JOBNAME.e$JOBID`
Description	Defines the path to be used for the standard error stream of the batch job.
Example	`> msub -e test12/stderr.txt` *The STDERR stream of the job will be placed in the relative (to execution) directory specified.*

-E	
Name	Environment Variables
Description	Moab adds the following variables, if populated, to the job's environment: • MOAB_ACCOUNT — Account name. • MOAB_BATCH — Set if a batch job (non-interactive). • MOAB_CLASS — Class name. • MOAB_DEPEND — Job dependency string. • MOAB_GROUP — Group name. • MOAB_JOBARRAYINDEX —For a job in an array, the index of the job. • MOAB_JOBARRAYRANGE — For a system with job arrays, the range of all job arrays. • MOAB_JOBID — Job ID. If submitted from the grid, grid jobid. • MOAB_JOBNAME — Job name. • MOAB_MACHINE — Name of the machine (i.e., destination resource manager) that the job is running on. • MOAB_NODECOUNT — Number of nodes allocated to job. • MOAB_NODELIST — Comma-separated list of nodes (listed singly with no ppn information). • MOAB_PARTITION — Partition name the job is running in. If grid job, cluster scheduler's name. • MOAB_PROCCOUNT — Number of processors allocated to job. • MOAB_QOS — QOS name. • MOAB_SUBMITDIR — Directory from which the job was submitted. • MOAB_TASKMAP — Node list with procs per node listed. <nodename>.<procs> • MOAB_USER — User name. This feature only works with Torque/PBS.
Example	`> msub -E mySim.cmd` *The job `mySim` will be submitted with extra environment variables.*

-F	
Name	Script Flags
Format	`"<STRING>"`

-F	
Description	Specifies the flags Torque will pass to the job script at execution time. The `-F` flag is only compatible with Torque resource managers.
Example	`> msub -F "arg1 arg2" -l nodes=1,walltime=60 files/job.sh` *Torque will pass parameters `arg1` and `arg2` to the `job.sh` script when the job executes.*

-h	
Name	Hold
Description	Specifies that a user hold be applied to the job at submission time.
Example	`> msub -h cmd.ll` *The job will be submitted with a user hold on it.*

-I	
Name	Interactive
Description	Declares the job is to be run interactively. `qsub` must exist on the same host as `msub` if the interactive job is destined for a Torque cluster, because the interactive `msub` request will be converted to a `qsub -I` request.
Example	`> msub -I job117.sh` *The job will be submitted in interactive mode.*

-j	
Name	Join
Format	`[eo|oe|n]`

-j	
Default	n (not merged)
Description	If `eo` is specified, the error and output streams are merged into the *error* stream. If `oe` is specified, the error and output streams will be merged into the *output* stream. If using either the -e or the -o option and the `-j eo|oe` option, the `-j` option takes precedence and all standard error and output messages go to the chosen output file.
Example	`> msub -j oe cmd.sh` *STDOUT and STDERR will be merged into one file.*

-k	
Name	Keep
Format	`[e|o|eo|oe|n]`
Default	n (not retained)
Description	Defines which (if either) of output and error streams will be retained on the execution host (overrides path for stream).
Example	`> msub -k oe myjob.sh` *STDOUT and STDERR for the job will be retained on the execution host.*

-K	
Name	Continue Running
Format	N/A
Description	Tells the client to continue running until the submitted job is completed. The client will query the status of the job every 5 seconds. The time interval between queries can be specified or disabled via the parameter MSUBQUERYINTERVAL.

-K	
	Use the `-K` option sparingly (if at all) as it slows down the Moab scheduler with frequent queries. Running ten jobs with the `-K` option creates an additional fifty queries per minute for the scheduler.

Example	`> msub -K newjob.sh`
	`3`
	`Job 3 completed*`
	**Only shows up after job completion.*

-l	
Name	Resource List

Format	`<STRING>`
	`-l [BANDWIDTH|DDISK|DEADLINE|DEPEND|DMEM|EXCLUDENODES|FEATURE...|]`
	Additional options can be referenced in 11.3 Resource Manager Extensions.

Description	Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. Resources native to the resource manager, scheduler resource manager extensions, or job flags can be specified. Note that resource lists are dependent on the resource manager in use. For information on specifying multiple types of resources for allocation, see 9.7.2 Multi-Req Support. Moab does not support the combination of *`msub -l excludenodes`* and *`ENABLEHIGHTHROUGHPUT TRUE`*.
Example	`> msub -l nodes=32:ppn=2,pmem=1800mb,walltime=3600,VAR=testvar:myvalue cmd.sh` *The job requires `32` nodes with `2` processors each, `1800` MB per task, a walltime of `3600` seconds, and a variable named `testvar` with a value of `myvalue`.* If the parameter JOBNODEMATCHPOLICY is not set, Moab does not reserve the requested number of processors on the requested number of nodes. It reserves the total number of requested processors (nodes x ppn) on any number of nodes. Rather than setting `nodes=<value>:ppn=<value>`, set `procs=<value>`, replacing `<value>` with the total number of processors the job requires. Note that `JOBNODEMATCHPOLICY` is not set by default.

-l	
	`> msub -l nodes=32:ppn=2 -l advres=!<resvid>` *This entry tells Moab to only consider resources other than the specified `<reservation id>`.*

-L	
Name	NUMA req_information
Description	This uses a different syntax than the -l resource_list option. Defines the NUMA-aware resource requests for NUMA hardware. This option will work with non-NUMA hardware. See '-L NUMA Resource Request' in the *Torque Resource Manager Administrator Guide* for the syntax and values.

-m	
Name	Mail Options
Format	`<STRING>` (either `n` or one or more of the characters `a`, `b`, and `e`)
Description	Defines the set of conditions (abort,begin,end) when the server will send a mail message about the job to the user.
Example	`> msub -m be cmd.sh` *Mail notifications will be sent when the job begins and ends.*

-M	
Name	Mail List
Format	`<user>[@<host>][,<user>[@<host>],...]`
Default	$JOBOWNER
Description	Specifies the list of users to whom mail is sent by the execution server. Overrides the `EMAILADDRESS` specified on the USERCFG credential.

-M	
Example	`> msub -M jon@node01,bill@node01,jill@node02 cmd.sh` *Mail will be sent to the specified users if the job is aborted.*

-n	
Name	Node Exclusive
Description	Allows a user to specify an exclusive-node access/allocation request for the job. See the node access policy SINGLEJOB for more information.
Example	`> msub -n job1187.sh` *Job will have exclusive access to each node on which it runs.*

-N	
Name	Name
Format	`<STRING>`
Default	STDIN or name of job script.
Description	Specifies the user-specified job name attribute.
Example	`> msub -N chemjob3 cmd.sh` *Job will be associated with the name* `chemjob3`*.*

-o	
Name	Output Path
Format	`[<hostname>:]<path>` - %J and %I are acceptable variables. %J is the master array name and %I is the array member index in the array.
Default	`$SUBMISSIONDIR/$JOBNAME.o$JOBID`
Description	Defines the path to be used for the standard output stream of the batch job.

-o	
	More variables are allowed when they are used in the job script instead of *msub -o*. In the job script, specify a `#PBS -o` line and input your desired variables. The allowable variables are: • OID • OTYPE • USER • OWNER • JOBID • JOBNAME Submitting a job script that has the line `#PBS -o $(USER)_$(JOBID)_$(JOBNAME).txt` results in a file called `<username>_<jobID>_<jobName>.txt`. Do not use *msub -o* when submitting a job script that has a `#PBS -o` line defined.

Example	`> msub -o test12/stdout.txt`
	The STDOUT stream of the job will be placed in the relative (to execution) directory specified.
	`> msub -t 1-2 -o /home/jsmith/simulations/%J-%I.out ~/sim5.sh`
	A job array is submitted and the name of the output files includes the master array index and the array member index.

-p	
Name	Priority
Format	`<INTEGER>` (between -1024 and 0)
Default	0
Description	Defines the priority of the job. To enable priority range from -1024 to +1023, see the parameter ENABLEPOSUSERPRIORITY.
Example	`> msub -p 25 cmd.sh` *The job will have a user priority of 25.*

-P	
Name	Proxy User
Format	`<user>[:<group>]`
Description	Allows a root user or manager to submit a job as another user. Moab treats proxy jobs as though the jobs were submitted by the supplied username. This option can only be used by users in the ADMINCFG[1] security level.
Example	`msub -P user1 cmd.pbs`

-q	
Name	Destination Queue (Class)
Format	`[<queue>][@<server>]`
Default	`[<DEFAULT>]`
Description	Defines the destination of the job. If no destination queue is specified and the environment variable MOAB_DEFAULTQUEUE is present, msub will use the environment variable when submitting the job.
Example	`> msub -q priority cmd.sh` *The job will be submitted to the priority queue.*

-r	
Name	Rerunable
Format	`[y|n]`
Default	n
Description:	Declares whether the job is rerunable.

-r	

Example	`> msub -r n cmd.sh`
	The job cannot be rerun.
	The default for qsub -r is 'y' (yes), which is opposite from msub -r. For better clarity, use the following instead:
	`msub -l [flags|jobflags]=restartable]`

-S	
Name	Shell
Format	`<path>[@<host>][,<path>[@<host>],...]`
Default	$SHELL
Description	Declares the shell that interprets the job script.
Example	`> msub -S /bin/bash` *The job script will be interpreted by the `/bin/bash` shell.*

-t	
Name	Job Arrays
Format	`<name>[<indexlist>]%<limit>`
Description	Starts a job array with the jobs in the index list. The limit variable specifies how many jobs can run at a time. For more information, see Submitting Job Arrays. Moab enforces an internal limit of 100,000 subjobs that a single array job submission can specify.
Example	`> msub -t myarray[1-1000]%4`

-u	
Name	User List
Format	`<user>[@<host>[,<user>[@<host>],...]`
Default	UID of *msub* command.
Description	Defines the user name under which the job is to run on the execution system.
Example	`> msub -u bill@node01 cmd.sh` *On `node01` the job will run under `Bill`'s UID, if permitted.*

-v	
Name	Variable List
Format	`<string>[,<string>,...]`
Description	Retrieves the values of the included environment variables on the job submission node (if no value is provided) or defines a name and value and exports these variables to the job's compute node(s).

Example	`> msub -v DEBUG cmd.sh`
	The `DEBUG` environment variable on the job submission node will be defined for the job.
	`> msub -v VAR1=xxx cmd.sh`
	The `VAR1` environment variable will be defined for the job, with a value of `xxx`.

-V	
Name	All Variables
Description	Declares that all environment variables in the msub environment are exported to the batch job.
Example	`> msub -V cmd.sh` *All environment variables will be exported to the job.*

-w	
Name	Working Directory
Format	`<path>`
Description	Defines the working directory path to be used for the job. If the -w option is not specified, the default working directory is the current directory. This option sets the environment variable PBS_O_WORKDIR.
Example	`> msub -l -w /tmp`

-W	
Name	Additional Attributes
Format	`<string>`
Description	Allows for specification of additional job attributes (see 11.3 Resource Manager Extensions).

Example	`> msub -W x=GRES:matlab:1 cmd.sh`
	The job requires one resource of `matlab`.
	This flag can be used to set a filter for what namespaces will be passed from a job to a trigger using a comma-delimited list. This limits the trigger's action to objects contained in certain workflows. For more information, see 17.4.1.E Requesting Name Space Variables.
	`> msub -W x="trigns=vc1,vc2"`
	The job passes namespaces `vc1` *and* `vc2` *to triggers.*

-x	
Format	`<script>` or `<command>`
Description	When running an interactive job, the `-x` flag makes it so that the corresponding script won't be parsed for PBS directives, but is instead a command that is launched once the interactive job has started. The job terminates at the completion of this command. This option works only when using Torque.

-x	
	The `-x` option for msub differs from qsub in that qsub does not require the script name to come directly after the flag. The msub command requires a script or command immediately after the `-x` declaration.

Example	`> msub -I -x ./script.pl`
	`> msub -I -x /tmp/command`

-z	
Name	Silent Mode
Description	The job's identifier will not be printed to stdout upon submission.
Example	`> msub -z cmd.sh` *No job identifier will be printout the stdout upon successful submission.*

Staging Data

Data staging, or the ability to copy data required for a job from one location to another or to copy resulting data to a new location (see Chapter 24: Data Staging for more information), must be specified at job submission. To stage data in, you use the *msub* `--stagein` and/or `--stageinfile` option, optionally with `--stageinsize`. You use similar options the same way for staging out: `--stageout`, `--stageoutfile`, and `--stageoutsize`. `--stagein` and `--stageout`, which you can use multiple times in the same *msub* command, allow you to specify a single file or directory to stage in or out. `--stageinfile` and `--stageoutfile` allow you to specify a text file that lists the files to stage in or out. The `--stageinsize` and [`--stageoutsize`] options allow you to estimate the total size of the files and directories that you want to stage in or out, which can help Moab make an intelligent guess about how long it will take to stage the data in or out, therefore ensuring that the job can start as soon as possible after the staging has occurred.

Staging a File or Directory

The `--stagein` and `--stageout` options use the same format.

`--<stagein|stageout><=| ><source>%<destination>`

Where `<source>` and `<destination>` take on the following format:
`[<user>@]<host>:/<path>[/<fileName>]`

Specifying a user and file name are optional. If you do not specify a file name, Moab will assume a directory.

```
> msub ... --stagein=student@biology:/stats/file001%admin@moab:/tmp/staging
<jobScript>
```

This *msub* commands tells Moab that the job requires `file001` from `student`'s `stats` directory on the `biology` server to be staged to `admin`'s `staging` directory on the `moab` server prior to the job's starting.

You can specify the option multiple times for the same *msub* command; however, staging large number of files is easier with `--stageinfile` or `--stageoutfile`.

You can also use `#MSUB` or `#PBS` within a job script to specify data staging options. For example:

```
#MSUB --stageinsize=1gb
#MSUB --stagein=...
```

See the section Sample User Job Script for more information. Note that the data staging options are not compatible with *qsub*.

Staging Multiple Files or Directories

The `--stageinfile` and `--stageoutfile` options use the same format. You must include the path to a text file that lists each file to stage in or out on its own line. Each file specification follows the same format as a `--stagein` or `--stageout` specification as described above.

The format of the command options looks like this: `--<stageinfile|stageoutfile><=| ><path>/<fileName>`

The file contains multiple lines with the following format:

`[<user>@]<host>:/<path>[/<fileName>]%[<user>@]<host>:/<path>[/<fileName>]`

`...`

Moab ignores blank lines in the file. You can comment out lines by preceding them with a pound sign (#). The following examples demonstrate what the `--stageinfile` option looks like on the command line and what the file it specifies might look like.

```
> msub ... --stageinfile=/tmp/myStagingFile <jobScript>
```

`/tmp/myStagingFile`:

```
student@biology:/stats/file001%moab:/tmp/staging
student@biology:/stats/file002%moab:/tmp/staging
student@biology:/stats/file003%moab:/tmp/staging
#student@biology:/stats/file004%moab:/tmp/staging
student@biology:/stats/file005%moab:/tmp/staging
```

```
student@biology:/stats/file006%moab:/tmp/staging
student@biology:/stats/file007%moab:/tmp/staging
student@biology:/stats/file008%moab:/tmp/staging
student@biology:/stats/file009%moab:/tmp/staging
student@biology:/stats/file010%moab:/tmp/staging
```

Moab stages in each file listed in `myStagingFile` to the `/tmp/staging` directory. Each file resides on the `biology` host as the `student` user. Moab ignores the blank line and the line specifying `file004`.

Stage in or out File Size

The optional `--stageinsize` and `--stageoutsize` options give you the opportunity to estimate the size of the file(s) or directory(-ies) being staged to aid Moab in choosing an appropriate start time.

Both options use the same format: `--<stageinsize|stageoutsize>=<integer> [unit]`

The integer indicates the size of the file(s) and directory(-ies) in megabytes unless you specify a different unit. Moab accepts the following case-insensitive suffixes: `KB`, `MB`, `GB`, or `TB`.

Moab copies the `/davidharris/research/recordlist` file, which is approximately 100 megabytes, from the biology node to the host where the job will run prior to job start:

```
> msub --stageinfile=/stats/file003 --stageinsize=100 <jobScript>
```

Moab copies all files specified in the `/davidharris/research/recordlist` file, which add up to approximately 1 gigabyte, to the host where the job will run prior to job start:

```
> msub --stageinfile=/stats/file002 --stageinsize=1gb <jobScript>
```

Return all the Job IDs in the Workflow at Submission Time

By default, *msub* will print the job ID to stdout at the time of submission. If you want *msub* to print all of the jobs that are created as part of the workflow template, you can use the *msub* `--workflowjobids` option to show all the job IDs at submission time:

```
$ echo sleep 60 | msub -l walltime=15 --workflowjobids

MoabA.3.dsin MoabA.3 MoabA.3.dsout
```

Job Script

The *msub* command supports job scripts written in any one of the following languages:

Language	Notes
PBS/Torque Job Submission Language	---
SSS XML Job Object Specification	---

Low Latency

The *msub* can be configured to return a job ID very quickly by eliminating the processing of some job attributes, filters, remap classes, job arrays, templates, workflows, limits and other information when a job is submitted. This can be done globally by configuring DISPLAYFLAGS USENOBLOCKMSUB or on the individual job submission by appending "--noblock" to the command line.

We recommend that when using a non-blocking msub, that the resource manager attribute JOBIDFORMAT be configured (and the resource manager flag PROXYJOBSUBMISSION if desired).

3.7.28.E /etc/msubrc

Sites that want to automatically add parameters to every job submission can populate the file /etc/msubrc with global parameters that every job submission will inherit.

For example, if you want every job to request a particular generic resource, the following /etc/msubrc could be used:

```
-W x=GRES:matlab:2
```

3.7.28.F Usage Notes

msub is designed to be as flexible as possible, allowing users accustomed to PBS or LoadLeveler syntax, to continue submitting jobs as they normally would. It is not recommended that different styles be mixed together in the same *msub* command.

When only one resource manager is configured inside of Moab, all jobs are immediately staged to the only resource manager available. However, when multiple resource managers are configured Moab will determine which resource manager can run the job soonest. Once this has been determined, Moab will stage the job to the resource manager.

It is possible to have Moab take a best effort approach at submission time using the forward flag. When this flag is specified, Moab will do a quick check and make an intelligent guess as to which resource manager can run the job soonest and then immediately stage the job.

Moab can be configured to instantly stage a job to the underlying resource manager (like Torque/LOADLEVELER) through the parameter INSTANTSTAGE. When set inside `moab.cfg`, Moab will migrate the job instantly to an appropriate resource manager. Once migrated, Moab will destroy all knowledge of the job and refresh itself based on the information given to it from the underlying resource manager.

In most instances Moab can determine what syntax style the job belongs to (PBS or LoadLeveler); if Moab is unable to make a guess, it will default the style to whatever resource manager was configured at compile time. If LoadLeveler and PBS were both compiled then LoadLeveler takes precedence.

Moab can translate a subset of job attributes from one syntax to another. It is therefore possible to submit a PBS style job to a LoadLeveler resource manager, and vice versa, though not all job attributes will be translated.

3.7.28.G Example

```
> msub -l nodes=3:ppn=2,walltime=1:00:00,pmem=100kb script2.pbs.cmd
4364.orion
```

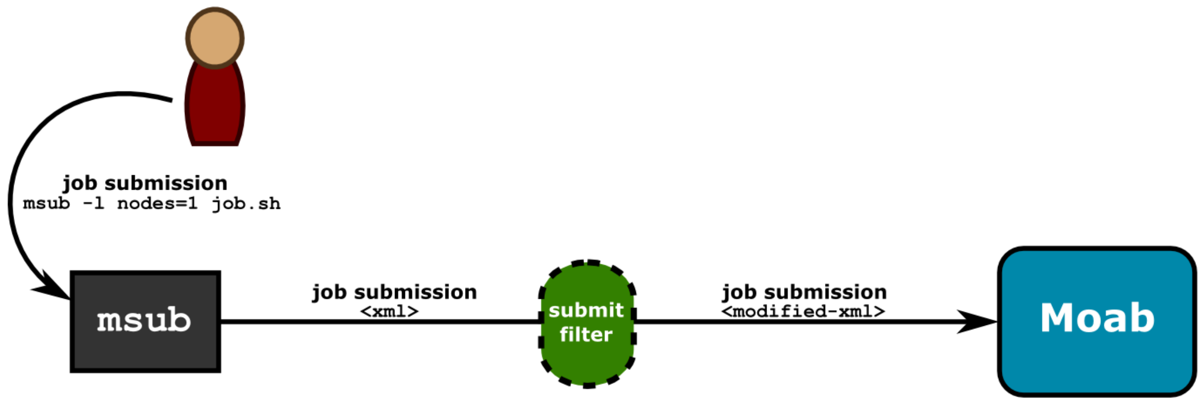
This example is the XML-formatted version of the above example. See the section Submitting Jobs Via msub in XML below for more information.

```
<job>
  <InitialWorkingDirectory>/home/user/test/perlAPI
  </InitialWorkingDirectory>
  <Executable>/home/user/test/perlAPI/script2.pbs.cmd
  </Executable>
  <SubmitLanguage>PBS</SubmitLanguage>
  <Requested>
    <Feature>ppn2</Feature>
    <Processors>3</Processors>
    <WallclockDuration>3600</WallclockDuration>
  </Requested>
</job>
```

3.7.28.H Applying the msub Submit Filter

When you use msub to submit a job, *`msub`* processes the input, converts it to XML, and sends the job specification XML to the Moab scheduler. You can create a submission filter to modify the job XML based on the criteria you set before Moab receives and processes it.

Image 3-1: Job submission process



The filter gives you the ability to customize the submission process, which is helpful if jobs should have certain defaults assigned to them, if you want to keep detailed submission statistics, or if you want to change job requests based on custom needs.

The submit filter, is a simple executable or script that receives XML via its standard input and returns the modified XML in its standard output. It modifies the attributes of the job specification XML based on policies you specify. It can perform various other actions at your request, too; for instance, logging. Once the submit filter has modified the job XML based on your criteria, it writes the XML representing the actual job submission to stdout. The new XML could potentially match the original XML, depending on whether the job met the criteria for modification set in the job submit filter script. Job submissions you want to proceed will leave the filter with an exit code of 0 and continue to Moab for scheduling. If the job meets the filter's specified criteria for rejection, it exits with a non-zero value, aborting the job submission process. You can configure the filter script to write a descriptive rejection message to stderr.

Job submit filters follow these rejection rules: 1) `msub` will reject job XML with an exit code of anything other than zero, 2) the `msub` command displays filter's error output on the command line, 3) `msub` will reject the job if the filter outputs invalid job XML, and 4) `msub` will reject the job if it violates any policies in your general Moab configuration; you cannot use a submit filter to bypass other policies.

To see the schema for job submission XML, refer to the section Submitting Jobs Via msub in XML below.

Submit Filter Types

You can implement submit filters on either the client or server side of a job submission. The primary differences between the two submit filter types are the location from which the filter runs, the powers and privileges of the user running the filter, and whether a user can bypass the filter. Client-based submit filters run from the `msub` client as the user who submits the job and can be bypassed, and server-based submit filters run from the Moab server as the user as which the server is running and cannot be bypassed.

Client-Based Submit Filter

Client-based filters run from the *msub* client as the user who is submitting the job. Because they do not have elevated privileges, the risk of client-based submit filters' being abused is low; however, it is possible for the client to specify its own configuration file and bypass the filter or substitute its own filter. Job submissions do not even reach the server if a client-based submit filter rejects it.

To configure *msub* to use the submit filter, give each submission host access to the submit filter script and add a SUBMITFILTER parameter to the Moab configuration file (`moab.cfg`) on each submission host. The following example demonstrates how you might modify the `moab.cfg` file:

```
SUBMITFILTER /home/submitfilter/filter.pl
```

If you experience problems with your submit filter and want to debug its interaction with *msub*, enter `msub --loglevel=9`. This will cause *msub* to print verbose log messages to the terminal.

Server-Based Submit Filter

Server-based submit filters run from the Moab server as the user as which the server is running. Because it runs as a privileged user, you must evaluate the script closely for security implications. A client configuration cannot bypass the filter.

To configure Moab to automatically apply a filter to all job submissions, use the SERVERSUBMITFILTER parameter. `SERVERSUBMITFILTER` specifies the path to a global job submit filter script, which Moab will run on the head node and apply to every job submitted.

```
SERVERSUBMITFILTER /opt/moab/scripts/jobFilter.pl
```

Moab runs `jobFilter.pl`, located in the `/opt/moab/scripts` directory, on the head node, applying the filter to all jobs submitted.

OutputFormat XML Tag

The 'OutputFormat' element is used by a job submit filter to alter the output of the msub command when it reports the submitted job's job ID. For example, if a job submit filter performs a complex procedure on behalf of the user, such as submitting system jobs for a predefined workflow to accomplish some function, the filter can set this element to a value that permits it to return the job IDs of the system jobs it submitted in addition to the user's job ID the msub command returns.

To illustrate this element's functionality using the Moab/DataWarp integration example, a DataWarp job submit filter submits a 'DataWarp instance creation/input data staging' script as a system job and a corresponding 'output data staging/DataWarp instance destruction' script as another system job, and then ties them together with job dependencies in a 'DataWarp job workflow' that causes the user job's execution to depend on the successful completion of the DataWarp creation/input staging job and the DataWarp output staging/DataWarp Destruction system job to depend on the user job, regardless whether it completes successfully or not, or is canceled. This DataWarp 3-job workflow guarantees the proper creation and destruction of job-based DataWarp storage; all set up and accomplished by a job submit filter.

However, users often create job workflows that have dependencies between their own jobs and may require the job IDs of all jobs to be made available in order to build a desired job workflow (i.e., 'jobB' may require 'jobA' to complete before 'jobB' is able to run). For example, if jobA was a DataWarp job and jobB should not run unless JobA successfully completes, but not until JobA's output data files are successfully staged, jobB must depend on jobA's job ID, and jobA's 'output data staging/DataWarp instance destruction' system job's job ID.

The user can indicate jobB's job dependencies when jobA is a DataWarp job using the job submission option: `-l depend=afterok:<jobAid>:<jobAoutputSystemJobId>`.

The OutputFormat XML tag provides a way for a job submit filter to pass the job IDs of additional jobs it submitted to perform a service on behalf of the user's job.

The <OutputFormat> tag must be added to the job tag. If it is outside, it is treated as an invalid XML.

For example, you might submit a job and a job submit filter submits two additional jobs to assist it; the first additional job, 'job11', will run before your job, and the second additional job, 'job12', needs to run after your job finishes. If the job submit filter requires them to output in the order of 'pre', 'user', and 'post' job IDs (which is the same order Moab outputs job IDs for user jobs with input and output data-staging options), it returns the following OutputFormat element as the user's job ID string:

```
<OutputFormat>moab.11 %s moab.12</OutputFormat>
```

msub displays the user ID string as "Moab.11 Moab.13 Moab.12"

This means that you can have all three job IDs delivered to the end user, or a job workflow generation script in an easy to read format.

Sample Submit Filter Script

The following example is a trivial implementation that will not affect whether a job is submitted. Use it as reference to verify that you are writing your filter properly.

```
#!/usr/bin/perl
use strict;
```

```
## Simple filter example that re-directs the output to a file.

my $file = "xmllog.out";

open FILE,">>$file" or die "Couldn't open $file: $!";
while (<>)
{
print FILE;
print;
}
close FILE;
```

3.7.28.I Submitting Jobs Via msub in XML

The following describes the XML format used with the *msub* command to submit a job to a Moab server. This information can be used to implement a filter and modify the XML normally generated by the *msub* command. The XML format described below is based on a variant of the Scalable Systems Software Job Object Specification.

Overall XML Format

The overall format of an XML request to submit a job can be shown through the following example:

```
<job>
**job attribute children**
</job>
```

An example of a simple job element with all the required children for a job submission is as follows:

```
<job>
    <Owner>user</Owner>
    <UserId>user</UserId>
    <GroupId>group</GroupId>
    <InitialWorkingDirectory>/home/user/directory</InitialWorkingDirectory>
    <UMask>18</UMask>
    <Executable>/full/path/to/script/or/first/line/of/stdin</Executable>
    <SubmitLanguage>Resource Manager Type</SubmitLanguage>
    <SubmitString>\START\23!/usr/bin/ruby\0contents\20of\20script</SubmitString>
  </job>
```

The section that follows entitled Job Element Format describes the possible attributes and their meanings in detail. In actuality, all that is needed to run a job in Moab is something similar to the following:

```
<job>
  <SubmitString>\START\23!/bin/sh\0asleep\201000</SubmitString>
</job>
```

This piece of XML requests Moab to submit a job using the contents of the SubmitString tag as a script, which is in this case a simple sh script to sleep for 1000 seconds. The *msub* command will create default values for all other needed attributes.

Job Element Format

The job element of the submission request contains a list of children and string values inside the children that represent the attribute/value pairs for the job. The earlier section, Overall XML Format, gives an example of this format. This section explains these attributes in detail.

Arguments — The arguments to be passed to the program are normally specified as arguments after the first argument specifying the script to be executed.

EligibleTime — The minimum time after which the job is eligible. This is the equivalent of the `-a` option in *msub*. Format: `[[[[CC]YY]MM]DD]hhmm[.SS]`

Environment — The semicolon list of environment variables that are exported to the job (taken from the *msub* command environment). The `-V`*msub* flag, for example, adds all the environment variables present at the time *msub* is invoked. Environment variables are delimited by the `~rs;` characters. Following is an example of the results of the `msub -v arg1=1,arg2=2` command:

```
<Environment>arg1=1~rs;arg2=2~rs;</Environment>
```

ErrorFile — Defines the path to be used for the standard error stream of the batch job. This is equivalent to the `-e` flag in *msub*.

Executable — This is normally either the name of the script to be executed, or the first line of the script if it is passed to *msub* through standard input.

Extension — The resource manager extension string. This can be specified via the command line in a number of ways, including the -W x= directive. Some other requests, such as some extensions used in the `-l` flag, are also converted to an extension string. The element has the following format:

```
<Extension>x=extension</Extension>
```

See the section Using the Extension Element to Submit Triggers below for additional information on the extension element.

GroupId — The string name of the group of the user submitting the job. This will correspond to the user's primary group on the operating system.

Hold — Specifies that a user hold be applied to the job at submission time. This is the equivalent to the *msub* flag `-h`. It will have the form:

```
<Hold>User</Hold>
```

InitialWorkingDirectory — Specifies in which directory the job should begin executing. This is equivalent to the `-d` flag in the *msub* command.

```
<InitialWorkingDirectory>/home/user/directory</InitialWorkingDirectory>
```

Interactive — Specifies that the job is to be interactive. This is the equivalent of the `-I` flag in *msub*.

```
<Interactive>TRUE</Interactive>
```

JobName — The user-specified job name attribute. This is equivalent to the `-N` flag in *msub*.

NotificationList — The job states after which an email should be sent and also specifies the users to be emailed. This is the equivalent of the `-m` and `-M` options in *msub*.

```
<NotificationList URI=user1:user2>JobFail,JobStart,JobEnd</NotificationList>
```

In this example, the command `msub -m abe -M user1:user2` ran indicating that emails should be sent when a job fails, starts, or ends, and that they should be sent to `user1` and `user2`.

OutputFile — Defines the path to be used for the standard output stream of the batch job. This is the equivalent of the `-o` flag in *msub*.

Priority — A user-requested priority value. This is the equivalent to the *msub*`-p` flag.

ProjectId — Defines the account associated with the job. This is equivalent to the `-A`*msub* flag.

QueueName — The requested class of the job. This is the equivalent of the *msub*`-q` flag.

Requested — Specifies resources and attributes the job specifically requests and has the following form:

```
<Requested>
     <... requested attributes>
</Requested>
```

See the section dedicated to requestable attributes in this element.

RMFlags — Flags that will get passed directly to the resource manager on job submission. This is equivalent to any arguments listed after the `-l`*msub* flag.

```
<RMFlags>arg1 arg2 arg3</RMFlags>
```

ShellName — Declares the shell that interprets the job script. This is equivalent to the *msub* flag `-S`.

SubmitLanguage — Resource manager whose language the job is using. Use Torque to specify a Torque resource manager.

SubmitString — Contains the contents of the script to be run, retrieved either from an actual script or from standard input. This also includes all resource manager specific directives that may have been in the script already or added as a result of other command line arguments.

TaskGroup — Groups a set of requested resources together. It does so by encapsulating a Requested element. For example, the command `msub -l nodes=2+nodes=3:ppn=2` generates the following XML:

```
<TaskGroup>
    <Requested>
      <Processors>2</Processors>
      <TPN>2</TPN>
    </Requested>
  </TaskGroup>
  <TaskGroup>
    <Requested>
      <Processors>2</Processors>
    </Requested>
  </TaskGroup>
```

UserId — The string value of the user ID of the job owner. This will correspond to the user's name on the operating system.

Using the Extension Element to Submit Triggers

Use the Extension element to submit triggers. With the exception of certain characters, the syntax for trigger creation is the same for non-XML trigger submission. See 17.1 About Object Triggers for detailed information on triggers. The ampersand (`&`) and less than sign (`<`) characters must be replaced for the XML to be valid. The following example shows how the Extension element is used to submit multiple triggers (separated by a semicolon). Note that ampersand characters are replaced with `&amp;` in the example:

```
<Job>
  <UserId>user1</UserId>
  <GroupId>user1</GroupId>
  <Arguments>60</Arguments>
  <Executable>/bin/sleep</Executable>

<Extension>x=trig:AType=exec&amp;Action="env"&amp;EType=start;trig:AType=exec&amp;Acti
on="trig2.sh"&amp;EType=end</Extension>
  <Processors>3</Processors>
  <Disk>500</Disk>
  <Memory>1024</Memory>
  <Swap>600</Swap>
  <WallclockDuration>300</WallclockDuration>
  <Environment>PERL5LIB=/perl5:</Environment>
</Job>
```

Elements Found in Requested Element

The following describes the tags that can be found in the Requested sub-element of the job element in a job submission request.

Nodes — A list of nodes that the job requests to be run on. This is the equivalent of the –`lhosts=<host-list>`*`msub`* directive.

```
<Requested>
```

```
  <Nodes>
    <Node>n1:n2</Node>
  </Nodes>
</Requested>
```

In this example, the users requested the hosts `n1` and `n2` with the command `msub -l host=n1:n2`.

Processors — The number of processors requested by the job. The following example was generated with the command `msub -l nodes=5`:

```
<Requested>
    <Processors>5</Processors>
</Requested>
```

TPN — Tasks per node. This is generated using the ppn resource manager extensions. For example, from `msub -l nodes=3:ppn=2`, the following results:

```
  <Requested>
    <Processors>6</Processors>
    <TPN>2</TPN>
  </Requested>
```

WallclockDuration — The requested wallclock duration of the job. This attribute is specified in the Requested element.

```
<Requested>
  <WallclockDuration>3600</WallclockDuration>
</Requested>
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.20 mjobctl - command to view, modify, and cancel jobs
- 3.7.1 checkjob - command to view detailed information about the job
- 3.7.26 mshow - command to view all jobs in the queue
- MSUBQUERYINTERVAL parameter
- SUBMITFILTER parameter

3.7.29 mvcctl (Moab Virtual Container Control)

3.7.29.A Synopsis

mvcctl -a `<OType>`:`<OName>`[,`<OType>`:`<OName>`] `<name>`

mvcctl -c [<description>]

mvcctl -d <name>

mvcctl -m <ATTR>=VAL[,<ATTR>=<VAL>] <name>

mvcctl -q [<name>|ALL] [--xml][--blocking][--flags=fullxml]

mvcctl -r <OType>:<OName>[,<OType>:<OName>] <name>

mvcctl -x <action><name>

[--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]

3.7.29.B Overview

A virtual container (VC) is a logical grouping of objects with a shared variable space and applied policies. Containers can hold virtual machines, jobs, reservations, and nodes. Containers can also be nested inside other containers.

A VC can be owned by a user, group, or account. Users can only view VCs to which they have access. Level 1 administrators (Admin1) can view and modify all VCs. The owner can also be changed. When modifying the owner, you must also specify the owner type:

```
mvcctl -m OWNER=acct:bob myvc
```

Adding objects to VCs at submission: You associate jobs and reservations with a specified VC upon submission. For example:

- mrsvctl -c ... -H <VC>
- msub ... -W x="vc=<VC>"

The user who submits objects must have access to the VC or the command is rejected.

3.7.29.C FullXML flag

The FullXML flag will cause the *mvcctl -q* command to show VCs in a hierarchical manner. If doing a non-XML (plaintext) query, sub-VCs will be listed inside their parent VCs. Each VC will be indented more than its parent.

```
VC[vc1] (vc1)
      Owner: user:jason
      VCs:
      VC[vc2] (vc2)
           Owner: user:jason
           Jobs:  Moab.1
```

```
          Rsvs:  system.1
          VCs:
          VC[vc3] (vc3)
              Owner: user:jason
      VC[vc4] (vc4)
          Owner: user:jason
```

If doing an XML query, the XML for all sub-objects (VCs, but also reservations, jobs, etc.) will also be included in the VC:

```
<Data>
  <vcs>
     <vc CREATETIME="1460666817" CREATOR="tshaw" DESCRIPTION="vc1"
     NAME="vc1" OWNER="user:tshaw" VCS="vc2,vc4">
       <ACL aff="positive" cmp="%=" name="tshaw" type="USER" />
     </vc>
     <vc CREATETIME="1460666818" CREATOR="tshaw" DESCRIPTION="vc2"
     JOBS="moab.1" NAME="vc2" OWNER="user:tshaw" RSVS="system.2"
     VCS="vc3">
       <ACL aff="positive" cmp="%=" name="tshaw" type="USER" />
     </vc>
     <vc CREATETIME="1460666818" CREATOR="tshaw" DESCRIPTION="vc3"
     NAME="vc3" OWNER="user:tshaw">
       <ACL aff="positive" cmp="%=" name="tshaw" type="USER" />
     </vc>
     <vc CREATETIME="1460666818" CREATOR="tshaw" DESCRIPTION="vc4"
     NAME="vc4" OWNER="user:tshaw">
       <ACL aff="positive" cmp="%=" name="tshaw" type="USER" />
     </vc>
   </vcs>
</Data>
```

Note that the XML from the blocking and non-blocking commands may differ.

3.7.29.D Virtual Container Flags

The following table indicates available VC flags and associated descriptions. Note that the `Deleting`, `HasStarted`, and `Workflow` flags cannot be set by a user but are helpful indicators of status.

VC Flags	
DestroyObjects	When the VC is destroyed, any reservations and jobs in the VC are also destroyed. This is recursive, so any objects in sub-VCs are also destroyed. Nodes are not removed.
DestroyWhenEmpty	When the VC is empty, it is destroyed.
Deleting	Set by the scheduler when the VC has been instructed to be removed. Internal flag. Admins cannot set or clear this flag.

VC Flags	
HasStarted	This flag is set on a VC workflow where at least one job has started. Internal flag. Admins cannot set or clear this flag.
HoldJobs	This flag will place a hold on any job that is submitted to the VC while this flag is set. It is not applied for already existing jobs that are added into the VC. If a job with a workflow is submitted to the VC, all jobs within the workflow are placed on hold.
NoReleaseWhenScheduled	Prevents Moab from lifting the UserHold on the workflow when it is scheduled. This enables an approval method where an admin must release the hold manually before the service is allowed to start as scheduled.
Workflow	Designates this VC as a VC that is for workflows. This flag is set when generated by a job template workflow. Workflow jobs can only be attached to one workflow VC. Internal flag. Admins cannot set or clear this flag.

3.7.29.E Options

-a	
Format	*mvcctl -a*<OType>:<OName>[,<OType>:<OName>] <name> Where <OType> is one of JOB, RSV, NODE, or VC.
Description	Add the given object(s).

Example	`mvcctl -a JOB:Moab.45 vc13`
	`>>job 'Moab.45' added to VC 'vc13'`

-c	
Format	*mvcctl -c* [<description>]
Description	Create a VC. The VC name is auto-generated. We recommend that you supply a description; otherwise the description is the same as the auto-generated name.

-c	

Example	`mvcctl -c "Linux testing machine"`
	`>>VC 'vc13' created`

-d	
Format	*mvcctl -d*<lab01>
Description	Destroy the VC.

Example	`mvcctl -d vc13`
	`>>VC 'vc13' destroyed`

-m	
Format	*mvcctl -m*<ATTR>=VAL[,<ATTR>=<VAL>] <name>
Description	Modify the VC. Attributes are flags, owner, reqstarttime, reqnodeset, variables, and owner; note that only the owner can modify owner. Use reqstarttime when implementing guaranteed start time to specify when jobs should start. The reqnodeset attribute indicates the node set that jobs should run in that are submitted to a VC.

Example	`mvcctl -m variables+=HV=node8 vc13`
	`>>VC 'vc13' successfully modified`
	
	`mvcctl -m flags+=DESTROYWHENEMPTY vc1`
	`>>VC 'vc1' successfully modified`
	
	`mvcctl -m messages="\"This VC is for internal use, etc.\"" vc5`
	`>>VC 'vc5' successfully modified`

-q	
Format	*mvcctl -q* [<name>|ALL] [--xml][--blocking][--flags=fullxml]
Description	Query VCs.

-q	

Example	
	```
	mvcctl -q ALL
	VC[vc13] (Linux testing machine)
	Create Time: 1311027343    Creator: jdoe
	Owner: user:jdoe
	ACL:   USER=%=jdoe+;
	Jobs:  Moab.45
	Vars:  HV=node88
	Flags: DESTROYWHENEMPTY
	```

-r	
Format	*mvcctl -r*<OType>:<OName>[,<OType>:<OName>] <name> Where <OType> is one of JOB, RSV, NODE, or VC.
Description	Remove the given object(s) from the VC.

Example	
	```
	mvcctl -r JOB:Moab.45 vc13
	>>job 'Moab.45' removed from VC 'vc13'
	```

-x	
Format	*mvcctl -x*<action><name>
Description	Executes the given action on the VC.
Example	`mvcctl -x schedulevc vc1`

3.7.30 showbf

3.7.30.A Synopsis

showbf [-a *account*] [-A] [-c *class*] [-d *duration*] [-D] [-f *features*] [-g *group*] [-h] [-L] [-m [==|>|>=|<|<=] *memory*] [-n *nodecount*] [-p *partition*] [-q *qos*] [-r *processorcount*] [-u *user*] [-v] [--blocking] [--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]

3.7.30.B Overview

Shows what resources are available for immediate use.

The results Moab returns do not include resources that may be freed due to preemption.

This command can be used by any user to find out how many processors are available for immediate use on the system. It is anticipated that users will use this information to submit jobs that meet these criteria and therefore obtain quick job turnaround times. This command incorporates down time, reservations, and node state information in determining the available backfill window.

If specific information is not specified, *`showbf`* will return information for the user and group running but with global access for other credentials. For example, if `-q qos` is not specified, Moab will return resource availability information for a job as if it were entitled to access all QOS based resources (i.e., resources covered by reservations with a QOS based ACL), if `-c class` is not specified, the command will return information for resources accessible by any class.

The *`showbf`* command incorporates node configuration, node utilization, node state, and node reservation information into the results it reports. This command does not incorporate constraints imposed by credential based fairness policies on the results it reports.

3.7.30.C Access

By default, this command can be used by any user or admin.

3.7.30.D Parameters

Parameter	Description
ACCOUNT	Account name.
CLASS	Class/queue required.
DURATION	Time duration specified as the number of seconds or in [DD:]HH:MM:SS notation.
FEATURELIST	Colon separated list of node features required.
GROUP	Specify particular group.

Parameter	Description
MEMCMP	Memory comparison used with the `-m` flag. Valid signs are >, >=, ==, <=, and <.
MEMORY	Specifies the amount of required real memory configured on the node, (in MB), used with the `-m` flag.
NODECOUNT	Specify number of nodes for inquiry with `-n` flag.
PARTITION	Specify partition to check with `-p` flag.
PROCESSORCOUNT	Specify number of processors required.
QOS	Specify QOS to check with `-q` flag.
USER	Specify particular user to check with `-u` flag.

3.7.30.E Options

Option	Description
-a	Show resource availability information only for the specified account.
-A	Show resource availability information for all users, groups, and accounts. By default, *`showbf`* uses the default user, group, and account ID of the user issuing the command.
--blocking	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.
-c	Show resource availability only for the specified class.
-d	Show resource availability information for specified duration.
-D	Display current and future resource availability notation.
-f	Display availability for the specified colon-separated list of node features.
-g	Show resource availability information only for specified group.

Option	Description
-h	Help for this command.
-L	Enforce Hard limits when showing available resources.
-m	Allows user to specify the memory requirements for the backfill nodes of interest. It is important to note that if the optional MEMCMP and MEMORY parameters are used, they must be enclosed in single ticks (') to avoid interpretation by the shell. For example, enter `showbf -m '==256'` to request nodes with 256 MB memory.
-n	Show resource availability information for a specified number of nodes. That is, this flag can be used to force *`showbf`* to display only blocks of resources with at least this many nodes available.
-p	Show resource availability information for the specified partition.
-q	Show information for the specified QOS.
-r	Show resource availability for the specified processor count.
-u	Show resource availability information only for specified user.
-v	Displays verbose information.

3.7.30.F Examples

Example 3-32: showbf

A job requiring up to 2 processors could be submitted for immediate execution in partition `ClusterA` for any duration. Additionally, a job requiring 1 processor could be submitted for immediate execution in partition `ClusterB`. Note that by default, each task is tracked and reported as a request for a single processor.

```
> showbf
Partition     Tasks  Nodes   StartOffset      Duration       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL               3      3      00:00:00      INFINITY  11:32:38_08/23
ReqID=0
ClusterA          1      1      00:00:00      INFINITY  11:32:38_08/23
ReqID=0
ClusterB          2      2      00:00:00      INFINITY  11:32:38_08/23
ReqID=0
```

StartOffset is the amount of time remaining before resources will be available.

Example 3-33: showbf -r

The output verifies that a backfill window exists for jobs requiring a 3 hour runtime and at least 16 processors. Specifying job duration is of value when time based access is assigned to reservations (i.e., using the `SRCFG` `TIMELIMIT` ACL).

```
> showbf -r 16 -d 3:00:00
Partition     Tasks  Nodes    Duration    StartOffset    StartDate
---------     -----  -----    --------    -----------    ----------
ALL           20     20       INFINITY    00:00:00       09:22:25_07/23
```

Example 3-34: showbf -m

A resource availability window is requested for processors located only on nodes with at least 512 MB of memory:

```
> showbf -m ' =512'
Partition     Tasks  Nodes    Duration    StartOffset    StartDate
---------     -----  -----    --------    -----------    ----------
ALL           20     20       INFINITY    00:00:00       09:23:23_07/23
ClusterA      10     10       INFINITY    00:00:00       09:23:23_07/23
ClusterB      10     10       INFINITY    00:00:00       09:23:23_07/23
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.31 showq
- 3.7.4 mdiag

3.7.31 showq

3.7.31.A Synopsis

`showq` [-b] [-g] [-l] [-c|-i|-r] [-n] [-N] [-o] [-p *partition*] [-R *rsvid*] [-s] [-S] [-u] [-v] [-w *<CONSTRAINT>*] [--blocking] [--noblock] `[--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]`

3.7.31.B Overview

Displays information about active, eligible, blocked, and/or recently completed jobs. Since the resource manager is not actually scheduling jobs, the job ordering it displays is not valid. The *showq* command displays the actual job ordering under the Moab Workload Manager. When used without flags, this command displays all jobs in active, idle, and non-queued states.

3.7.31.C Access

By default, this command can be run by any user. However, the *-c*, *-i*, and *-r* flags can only be used by level 1, 2, or 3 Moab Admins.

3.7.31.D Options

Option	Argument	Description
-b	---	Display blocked jobs only.
-c	---	Display details about recently completed jobs (see the section Detailed Completed Job Report below and the parameter JOBCPURGETIME).
-g	---	Display grid job and system IDs for all jobs.
-i	---	Display extended details about idle jobs.
-l	---	Display local/remote view. For use in a Grid environment, displays job usage of both local and remote compute resources.
-n	---	Displays normal showq output, but lists job names under JOBID.
-N	---	Show the node/task allocation of the job.
-o	---	Displays jobs in the active queue in the order specified (uses format *showq -o <specifiedOrder>*). Options include REMAINING, REVERSEREMAINING, JOB, USER, STATE, and STARTTIME. The default is REMAINING.
-p	partition	Display only jobs assigned to the specified partition.

Option	Argument	Description
-r	---	Display extended details about active (running) jobs (see the section Detailed Completed Job Report below).
-R	rsvid	Display only jobs that overlap the specified reservation.
-s	---	Display workload summary.
-S	---	Display system jobs.
-u	---	Display all running jobs for a particular user.
-v	---	Display local and full resource manager job IDs, and partitions. If specified with the `-i` option, will display job reservation time. To see a summary of array subjobs, run `checkjob -v <jobID>`. To see array subjobs in showq, include the `--blocking` option.
-w	constraint	Display only jobs associated with the specified constraint. Valid constraints include user, group, jobgroup, acct, nodefeature, class, and qos (see showq -w examples below).
--blocking	---	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler. This option also causes *showq* to display an individual line for each array subjob.
--noblock	---	Use cache information for a faster response.

3.7.31.E Details

Beyond job information, the *showq* command will also report if the scheduler is stopped or paused or if a system reservation is in place. Further, the *showq* command will also report public system messages.

3.7.31.F Examples

- Default Report
  - Detailed Active/Running Job Report
  - Eligible Jobs
  - Detailed Completed Job Report
- Filtered Job Report

Default Report

The output of this command is divided into three parts: Active Jobs, Eligible Jobs, and Blocked Jobs.

```
> showq

active jobs------------------------
JOBIDUSERNAMESTATEPROCSREMAINING            STARTTIME

12941              sartois    Running    25      2:44:11  Thu Sep  1 15:02:50
12954               tgates    Running     4      2:57:33  Thu Sep  1 15:02:52
12944                eval1    Running    16      6:37:31  Thu Sep  1 15:02:50
12946               tgates    Running     2   1:05:57:31  Thu Sep  1 15:02:50

4 active jobs             47 of 48 processors active (97.92%)
                          32 of 32 nodes active      (100.00%)

eligible jobs----------------------
JOBID              USERNAME      STATE   PROCS     WCLIMIT            QUEUETIME

12956              cfosdyke       Idle     32     6:40:00  Thu Sep  1 15:02:50
12969              cfosdyke       Idle      4     6:40:00  Thu Sep  1 15:03:23
12939                 eval1       Idle     16     3:00:00  Thu Sep  1 15:02:50
12940               mwillis       Idle      2     3:00:00  Thu Sep  1 15:02:50
12947               mwillis       Idle      2     3:00:00  Thu Sep  1 15:02:50
12949                 eval1       Idle      2     3:00:00  Thu Sep  1 15:02:50
12953                tgates       Idle     10     4:26:40  Thu Sep  1 15:02:50
12955                 eval1       Idle      2     4:26:40  Thu Sep  1 15:02:50
12957                tgates       Idle     16     3:00:00  Thu Sep  1 15:02:50
12963                 eval1       Idle     16  1:06:00:00  Thu Sep  1 15:02:52
12964                tgates       Idle     16  1:00:00:00  Thu Sep  1 15:02:52
12937               allendr       Idle      9  1:00:00:00  Thu Sep  1 15:02:50
12962                aacker       Idle      6    00:26:40  Thu Sep  1 15:02:50
12968               tamaker       Idle      1     4:26:40  Thu Sep  1 15:02:52

14 eligible jobs

blocked jobs-----------------------
JOBID              USERNAME      STATE   PROCS     WCLIMIT            QUEUETIME


0 blocked jobs

Total jobs:  18
```

The fields are as follows:

Column	Description
JOBID	Job identifier.
USERNAME	User owning job.
STATE	Job State. Current batch state of the job.
PROCS	Number of processors being used by the job.
REMAINING/WCLIMIT	For active jobs, the time the job has until it has reached its wallclock limit or for idle/blocked jobs, the amount of time requested by the job. Time specified in [DD:]HH:MM:SS notation.
STARTTIME	Time job started running.

Active Jobs

Active jobs are those that are Running or Starting and consuming resources. Displayed are the job ID*, the job's owner, and the job state. Also displayed are the number of processors allocated to the job, the amount of time remaining until the job completes (given in HH:MM:SS notation), and the time the job started. All active jobs are sorted in 'Earliest Completion Time First' order.

*Job IDs can be marked with a single character to specify the following conditions:

Character	Description
_ (underbar)	job violates usage limit
*** (asterisk)**	job is backfilled AND is preemptible
+ (plus)	job is backfilled AND is NOT preemptible
- (hyphen)	job is NOT backfilled AND is preemptible

Detailed active job information can be obtained using the *-r* flag.

Eligible Jobs

Eligible Jobs are those that are queued and eligible to be scheduled. They are all in the Idle job state and do not violate any fairness policies or have any job holds in place. The jobs in the Idle section display the same information as the Active Jobs section except that the wallclock CPULIMIT is specified rather than job time REMAINING, and job QUEUETIME is displayed rather than job STARTTIME. The jobs in this section are ordered by job priority. Jobs in this queue are considered eligible for both scheduling and backfilling.

Detailed eligible job information can be obtained using the `-i` flag.

Blocked Jobs

Blocked jobs are those that are ineligible to be run or queued. Jobs listed here could be in a number of states for the following reasons:

State	Description
Idle	Job violates a fairness policy. Use `diagnose -q` for more information.
UserHold	A user hold is in place.
SystemHold	An administrative or system hold is in place.
BatchHold	A scheduler batch hold is in place (used when the job cannot be run because the requested resources are not available in the system or because the resource manager has repeatedly failed in attempts to start the job).
Deferred	A scheduler defer hold is in place (a temporary hold used when a job has been unable to start after a specified number of attempts. This hold is automatically removed after a short period of time).
NotQueued	Job is in the resource manager state NQ (indicating the job's controlling scheduling daemon in unavailable).

A summary of the job queue's status is provided at the end of the output.

Example 3-35: Detailed Active/Running Job Report

```
> showq -r

active jobs------------------------
JOBID               S  PAR  EFFIC  XFACTOR  Q      USER    GROUP        MHOST PROCS
REMAINING            STARTTIME

12941               R    3 100.00      1.0  -   sartois   Arches       G5-014 25
2:43:31  Thu Sep  1 15:02:50
12954               R    3 100.00      1.0 Hi    tgates   Arches       G5-016  4
2:56:54  Thu Sep  1 15:02:52
12944               R    2 100.00      1.0 De     eval1  RedRock     P690-016 16
6:36:51  Thu Sep  1 15:02:50
12946               R    3 100.00      1.0  -    tgates   Arches       G5-001  2
1:05:56:51  Thu Sep  1 15:02:50
```

```
4 active jobs              47 of 48 processors active (97.92%)
                           32 of 32 nodes active      (100.00%)

Total jobs:  4
```

After displaying the running jobs, a summary is provided indicating the number of jobs, the number of allocated processors, and the system utilization.

Column	Description
JOBID	Name of active job.
S	Job State. Either R for Running or S for Starting.
PAR	Partition in which job is running.
EFFIC	CPU efficiency of job.
XFACTOR	See the section Expansion Factor (XFACTOR) Subcomponent for a detailed description.
Q	Quality Of Service specified for job.
USER	User owning job.
GROUP	Primary group of job owner.
MHOST	Master Host running primary task of job.
PROCS	Number of processors being used by the job.
REMAINING	Time the job has until it has reached its wallclock limit. Time specified in HH:MM:SS notation.
STARTTIME	Time job started running.

```
> showq -i

eligible jobs----------------------
JOBID                 PRIORITY  XFACTOR  Q      USER    GROUP   PROCS     WCLIMIT
CLASS      SYSTEMQUEUETIME

12956*                      20      1.0  -  cfosdyke  RedRock      32     6:40:00
batch   Thu Sep  1 15:02:50
12969*                      19      1.0  -  cfosdyke  RedRock       4     6:40:00
batch   Thu Sep  1 15:03:23
```

```
12939                        16       1.0  -     eval1  RedRock     16      3:00:00
batch  Thu Sep  1 15:02:50
12940                        16       1.0  -   mwillis   Arches      2      3:00:00
batch  Thu Sep  1 15:02:50
12947                        16       1.0  -   mwillis   Arches      2      3:00:00
batch  Thu Sep  1 15:02:50
12949                        16       1.0  -     eval1  RedRock      2      3:00:00
batch  Thu Sep  1 15:02:50
12953                        16       1.0  -    tgates   Arches     10      4:26:40
batch  Thu Sep  1 15:02:50
12955                        16       1.0  -     eval1  RedRock      2      4:26:40
batch  Thu Sep  1 15:02:50
12957                        16       1.0  -    tgates   Arches     16      3:00:00
batch  Thu Sep  1 15:02:50
12963                        16       1.0  -     eval1  RedRock     16   1:06:00:00
batch  Thu Sep  1 15:02:52
12964                        16       1.0  -    tgates   Arches     16   1:00:00:00
batch  Thu Sep  1 15:02:52
12937                         1       1.0  -   allendr  RedRock      9   1:00:00:00
batch  Thu Sep  1 15:02:50
12962                         1       1.2  -    aacker  RedRock      6     00:26:40
batch  Thu Sep  1 15:02:50
12968                         1       1.0  -   tamaker  RedRock      1      4:26:40
batch  Thu Sep  1 15:02:52

14 eligible jobs

Total jobs:  14
```

The fields are as follows:

Column	Description
JOBID	Name of job.
PRIORITY	Calculated job priority.
XFACTOR	See the section Expansion Factor (XFACTOR) Subcomponent for a detailed description.
Q	Quality Of Service specified for job.
USER	User owning job.
GROUP	Primary group of job owner.
PROCS	Minimum number of processors required to run job.
WCLIMIT	Wallclock limit specified for job. Time specified in HH:MM:SS notation.
CLASS	Class requested by job.

Column	Description
SYSTEMQUEUETIME	Time job was admitted into the system queue.

An asterisk at the end of a job (job `12956*` in this example) indicates that the job has a job reservation created for it. The details of this reservation can be displayed using the checkjob command.

Example 3-36: Detailed Completed Job Report

```
> showq -c
completed jobs------------------------
JOBID                SCCODE  PAR  EFFIC  XFACTOR  Q  USERNAME    GROUP         MHOST
PROC    WALLTIME              STARTTIME
13098                C     0  bas  93.17      1.0  -   sartois   Arches        G5-014
25    2:43:31  Thu Sep  1 15:02:50
13102                C     0  bas  99.55      2.2 Hi    tgates   Arches        G5-016
 4    2:56:54  Thu Sep  1 15:02:52
13103                C     2  tes  99.30      2.9 De     eval1  RedRock     P690-016
16    6:36:51  Thu Sep  1 15:02:50
13115                C     0  tes  97.04      1.0  -    tgates   Arches        G5-001
 2 1:05:56:51  Thu Sep  1 15:02:50
3 completed jobs
```

The fields are as follows:

Column	Description
JOBID	job ID for completed job.
S	Job State. Either C for Completed, or V for Vacated.
CCODE	Completion code reported by the job.
PAR	Partition in which job ran.
EFFIC	CPU efficiency of job.
XFACTOR	See the section Expansion Factor (XFACTOR) Subcomponent for a detailed description.
Q	Quality of Service specified for job.
USERNAME	User owning job.

Column	Description
GROUP	Primary group of job owner.
MHOST	Master Host that ran the primary task of job.
PROCS	Number of processors being used by the job.
WALLTIME	Wallclock time used by the job. Time specified in [DD:]HH:MM:SS notation.
STARTTIME	Time job started running.

After displaying the active jobs, a summary is provided indicating the number of jobs, the number of allocated processors, and the system utilization.

If the DISPLAYFLAGS parameter is set to `ACCOUNTCENTRIC`, job group information will be replaced with job account information.

Example 3-37: Filtered Job Report

Show only jobs associated with user `john`, class `benchmark`, and nodefeature `bigmem`:

```
> showq -w class=benchmark -w user=john -w nodefeature=bigmem
...
```

Example 3-38: Filtered Job Report

Show only jobs associated with jobgroup `workflow1`:

```
> showq -w jobgroup=workflow1
...
```

3.7.31.G Job Array

Job arrays show the name of the job array and then in parenthesis, the number of subjobs in the job array that are in the specified state:

```
> showq

active jobs------------------------
JOBID              USERNAME      STATE  PROCS   REMAINING            STARTTIME

Moab.1(14)          aesplin    Running    14    00:59:41  Fri May 27 14:58:57

14 active jobs            14 of 14 processors in use by local jobs (100.00%)
2 of 2 nodes active      (100.00%)
```

```
eligible jobs----------------------
JOBID               USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME

Moab.1(4)            aesplin       Idle      4     1:00:00  Fri May 27 14:58:52

4 eligible jobs

blocked jobs-----------------------
JOBID               USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME

Moab.1(2)            aesplin    Blocked      2     1:00:00  Fri May 27 14:58:52

2 blocked jobs

Total jobs:  20
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.30 showbf - command to display resource availability.
- 3.7.4 mdiag - command to display detailed job diagnostics.
- 3.7.1 checkjob - command to check the status of a particular job.
- JOBCPURGETIME parameter to adjust the duration of time Moab preserves information about completed jobs
- DISPLAYFLAGS parameter to control what job information is displayed

3.7.32 showhist.moab.pl

3.7.32.A Synopsis

`showhist.moab.pl` [-a *accountname*]
[-c *classname*] [-e *enddate*]
[-g *groupname*] [-j *jobid*] [-n *days*]
[-q *qosname*] [-s *startdate*]
[-u *username*]

3.7.32.B Overview

The `showhist.moab.pl` script displays historical job information. Its purpose is similar to the checkjob command's, but `showhist.moab.pl` displays information about jobs that have already completed.

3.7.32.C Access

By default, this script's use is limited to admins on the head node; however, end users can also be given power to run the script. To grant access to the script to end users, move `showhist.moab.pl` from the `tools` directory to the `bin` directory.

3.7.32.D Options

-a (Account)	
Format	`<ACCOUNTNAME>`
Description	Displays job records matching the specified account.
Example	`> showhist.moab.pl -a myAccount` *Information about jobs related to the account `myAccount` is displayed.*

-c (Class)	
Format	`<CLASSNAME>`
Description	Displays job records matching the specified class (queue).
Example	`> showhist.moab.pl -c newClass` *Information about jobs related to the class `newClass` is displayed.*

-e (End Date)	
Format	`YYYY-MM-DD`
Description	Displays the records of jobs recorded before or on the specified date.

Example	`> showhist.moab.pl -e 2025-01-03`
	Information about all jobs recorded on or before January 3, 2025 is displayed.
	`> showhist.moab.pl -s 2025-01-01 -e 2025-01-31`
	Information is displayed about all jobs recorded in January 2025.

-g (Group)	
Format	`<GROUPNAME>`
Description	Displays job records matching the specified group.
Example	`> showhist.moab.pl -g admins` *Information about jobs related to the group* `admins` *is displayed.*

-j (Job ID)	
Format	`<JOBID>`
Description	Displays job records matching the specified job ID.
Example	`> showhist.moab.pl -j moab01` *Information about job* `moab01` *is displayed.*

-n (Number of Days)	
Format	`<INTEGER>`
Description	Restricts the number of past jobs to search by a specified number of days relative to today.
Example	`> showhist.moab.pl -n 90 -j moab924` *Displays job information for job* `moab924`*. The search is restricted to the last 90 days.*

-q (QoS)	
Format	`<QOSNAME>`
Description	Displays job records matching the specified quality of service.
Example	`> showhist.moab.pl -q myQos` *Information about jobs related to the QoS* `myQos` *is displayed.*

-s (Start Date)	
Format	`YYYY-MM-DD`
Description	Displays the records of jobs that recorded on the specified date and later.

Example	`> showhist.moab.pl -s 1776-07-04`
	Information about all jobs recorded on July 4, 1776 and later is displayed.
	`> showhist.moab.pl -s 2020-07-05 -e 2024-07-05`
	Information is displayed about all jobs recorded between July 5, 2020 and July 5, 2024.

-u (User)	
Format	`<USERNAME>`
Description	Displays job records matching the specified user.
Example	`> showhist.moab.pl -u bob` *Information about user `bob`'s jobs is displayed.*

Sample Output

```
> showhist.moab.pl
```

```
Job Id             : Moab.4
User Name          : user1
Group Name         : company
Queue Name         : NONE
Processor Count    : 4
Wallclock Duration: 00:00:00
Submit Time        : Mon Nov 21 10:48:32 2024
Start Time         : Mon Nov 21 10:49:37 2024
End Time           : Mon Nov 21 10:49:37 2024
Exit Code          : 0
Allocated Nodelist: 10.10.10.3

Job Id             : Moab.1
Executable         : 4
User Name          : user1
Group Name         : company
Account Name       : 1321897709
Queue Name         : NONE
Quality Of Service: 0M
Processor Count    : -0
Wallclock Duration: 00:01:05
Submit Time        : Mon Nov 21 10:48:29 2024
Start Time         : Mon Nov 21 10:48:32 2024
End Time           : Mon Nov 21 10:49:37 2024
Exit Code          : 0
Allocated Nodelist: 512M
```

Information is displayed for all completed jobs.

When a job's Start Time and End Time are the same, the job is infinite and still running.

Related Topics

- 3.7.1 checkjob - explains how to query for a status report for a specified job
- 3.7.9 mdiag -j - command to display additional detailed information regarding jobs
- 3.7.31 showq - command to show high-level job summaries

3.7.33 showres

3.7.33.A Synopsis

showres [-f] [-n [-g]] [-o] [-r] [-v] [reservationid] [--blocking] [--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]

3.7.33.B Overview

This command displays all reservations currently in place within Moab. The default behavior is to display reservations on a reservation-by-reservation basis.

3.7.33.C Access

By default, this command can be run by any Moab Admin.

3.7.33.D Options

Option	Description
-f	Show free (unreserved) resources rather than reserved resources. The `-f` flag cannot be used in conjunction with any other flag.
-g	When used with the `-n` flag, shows *`grep`*-able output with nodename on every line.
-n	Display information regarding all nodes reserved by `<RSVID>`.
-o	Display all reservations that overlap `<RSVID>` (in time and space). Not supported with `-n` flag.
-r	Display reservation timeframes in relative time mode.
-v	Show verbose output. If used with the `-n` flag, the command will display all reservations found on nodes contained in `<RSVID>`. Otherwise, it will show long reservation start dates including the reservation year.

Parameter	Description
RSVID	ID of reservation of interest — optional

3.7.33.E Examples

This example shows all reservations on the system:

```
> showres

ReservationID       Type S       Start         End     Duration     N/P     StartTime

12941                Job R   -00:05:01     2:41:39      2:46:40   13/25     Thu Sep  1
```

```
15:02:50
12944                Job R   -00:05:01     6:34:59     6:40:00   16/16   Thu Sep  1
15:02:50
12946                Job R   -00:05:01  1:05:54:59  1:06:00:00    1/2    Thu Sep  1
15:02:50
12954                Job R   -00:04:59     2:55:01     3:00:00    2/4    Thu Sep  1
15:02:52
12956                Job I  1:05:54:59  1:12:34:59     6:40:00   16/32   Fri Sep  2
21:02:50
12969                Job I     6:34:59    13:14:59     6:40:00    4/4    Thu Sep  1
21:42:50

6 reservations located
```

The fields are as follows:

Column	Description
Type	Reservation Type: Job or User.
ReservationID	The name of the reservation. Job reservation names are identical to the job name. User, Group, or Account reservations are the user, group, or account name followed by a number. System reservations are given the name SYSTEM followed by a number.
S	State. This field is valid only for job reservations. It indicates whether the job is (S)tarting, (R)unning, or (I)dle.
Start	Relative start time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time.
End	Relative end time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time. Reservations that will not complete in 1,000 hours are marked with the keyword INFINITY.
Duration	Duration of the reservation in HH:MM:SS notation. Reservations lasting more than 1,000 hours are marked with the keyword INFINITY.
Nodes	Number of nodes involved in reservation.
StartTime	Time reservation became active.

Example 3-39: showres -n

This shows reservations for nodes:

```
> showres -n
reservations on Thu Sep  1 16:49:59

NodeName         Type      ReservationID    JobState Task       Start     Duration
```

```
StartTime

G5-001           Job              12946    Running    2    -1:47:09  1:06:00:00  Thu
Sep  1 15:02:50
G5-001           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-002           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-002           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-003           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-003           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-004           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-004           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-005           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-005           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-006           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-006           Job              12953    Running    2   -00:29:37     4:26:40  Thu
Sep  1 16:20:22
G5-007           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-007           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-008           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-008           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-009           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-009           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-010           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-010           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-011           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-011           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-012           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-012           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-013           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-013           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-014           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-014           Job              12939    Running    2   -00:29:37     3:00:00  Thu
Sep  1 16:20:22
G5-015           Job              12956       Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-015           Job              12949    Running    2   -00:08:57     3:00:00  Thu
Sep  1 16:41:02
```

```
G5-016           Job              12956        Idle    2  1:04:12:51     6:40:00  Fri
Sep  2 21:02:50
G5-016           Job              12947     Running    2   -00:08:57     3:00:00  Thu
Sep  1 16:41:02
P690-001         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-002         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-003         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-004         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-005         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-006         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-007         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-008         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-009         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-010         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-011         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-012         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-013         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-013         Job              12969        Idle    1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-014         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-014         Job              12969        Idle    1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-015         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-015         Job              12969        Idle    1     4:52:51     6:40:00  Thu
Sep  1 21:42:50
P690-016         Job              12944     Running    1    -1:47:09     6:40:00  Thu
Sep  1 15:02:50
P690-016         Job              12969        Idle    1     4:52:51     6:40:00  Thu
Sep  1 21:42:50

52 nodes reserved
```

The fields are as follows:

Column	Description
NodeName	Node on which the reservation is placed.
Type	Reservation Type: Job or User.
ReservationID	The name of the reservation. Job reservation names are identical to the job name. User, Group, or Account reservations are the user, group, or account name followed by a number. System reservations are given the name

Column	Description
	SYSTEM followed by a number.
JobState	This field is valid only for job reservations. It indicates the state of the job associated with the reservation.
Start	Relative start time of the reservation. Time is displayed in HH:MM:SS notation and is relative to the present time.
Duration	Duration of the reservation in HH:MM:SS notation. Reservations lasting more than 1000 hours are marked with the keyword INFINITY.
StartTime	Time the reservation became active.

Example 3-40: showres

Information for a specific reservation (job) is displayed:

```
> showres 12956

ReservationID       Type S       Start         End     Duration     N/P     StartTime

12956                Job I  1:04:09:32  1:10:49:32      6:40:00   16/32     Fri Sep  2
21:02:50

1 reservation located
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.24 mrsvctl - create new reservations
- 3.7.24 mrsvctl - release existing reservations
- 3.7.4 mdiag - diagnose/view the state of existing reservations
- 6.1.1 Reservation Overview - description of reservations and their use

3.7.34 showstart

3.7.34.A Synopsis

`showstart` {*jobid*|*proccount*[@*duration*]|*s3jobspec*} [-e {all|hist|prio|rsv}] [-f] [-g [*peer*]] [-l qos=<*QOS*>] [--blocking] [`--format=xml`] [`-v`] `[--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]`

3.7.34.B Overview

This command displays the estimated start time of a job based a number of analysis types. This analysis may include information based on historical usage, earliest available reservable resources, and priority based backlog analysis. Each type of analysis will provide somewhat different estimates based on current cluster environmental conditions. The default estimation method used is determined by the value of the DEFAULTSTARTTIMEQUERY parameter, which defaults to PRIORITY.

showstart is a processor-intensive command. Multiple submissions per iteration may slow Moab's scheduling, especially on larger/busier systems.

The start time estimate Moab returns does not account for resources that will become available due to preemption.

showstart *only* determines where a job would run if it were to run next, taking into account all currently running jobs, queued idle jobs with advance reservations, and all current standing and administrative reservations in the system.

For example, assume `RESERVATIONDEPTH` is set to 1 (the default value), job 12300 is at the top of the idle queue and has an advance reservation to run next, and job 12312 is in position 12 in the idle queue. If the owner of job 12312 runs `showstart 12312`, in calculating where the job will run, Moab does not consider jobs 12301-12311. It only estimates where and when job 12312 would be scheduled to run after job 12300 starts.

Historical analysis utilizes historical queue times for jobs that match a similar processor count and job duration profile. This information is updated on a sliding window, which is configurable within `moab.cfg`

Reservation based start time estimation incorporates information regarding current administrative, user, and job reservations to determine the earliest time the specified job could allocate the needed resources and start running. In essence, this estimate will indicate the earliest time the job would start assuming this job was the highest priority job in the queue.

Priority based job start analysis determines when the queried job would fit in the queue and determines the estimated amount of time required to complete the jobs that are currently running or scheduled to run before this job can start.

In all cases, if the job is running, this command will return the time the job started. If the job already has a reservation, this command will return the start time of the reservation.

3.7.34.C Access

By default, this command can be run by any user.

3.7.34.D Parameters

Parameters	Description
DURATION	Duration of pseudo-job to be checked in format [[[DD:]HH:]MM:]SS (default duration is 1 second).
QOS	Specifies what QOS the job must start under, using the same syntax as the msub command. Currently, no other resource manager extensions are supported. This flag only applies to hypothetical jobs by using the proccount [@duration] syntax.
JOBID	Job to be checked.
PROCCOUNT	Number of processors in pseudo-job to be checked.
S3JOBSPEC	XML describing the job according to the Dept. of Energy Scalable Systems Software/S3 job specification.

3.7.34.E Options

Parameters	Description
--blocking	Do not use cache information in the output. The `--blocking` flag retrieves results exclusively from the scheduler.

Parameters	Description
-e	Estimate method. If not specified, Moab will use the value of the DEFAULTSTARTTIMEQUERY parameter, which defaults to PRIORITY.
-f	Use feedback. If specified, Moab will apply historical accuracy information to improve the quality of the estimate.

-g	Grid mode. Obtain showstart information from remote resource managers. If -g is not used and Moab determines that job is already migrated, Moab obtains showstart information from the remote Moab where the job was migrated to. All resource managers can be queried by using the keyword 'all', which returns all information in a table.
	`$ showstart -g all head.1`
	`Estimated Start Times`
	`[ Remote RM ] [ Reservation ] [ Priority ] [ Historical ]`
	`[ c1 ] [ 00:15:35 ] [ ] [ ]`
	`[ c2 ] [ 3:15:38 ] [ ] [ ]`

-l qos=<QOS>	Specifies what QOS the job must start under, using the same syntax as the `msub` command. Currently, no other resource manager extensions are supported. This flag only applies to hypothetical jobs by using the `proccount[@duration]` syntax.
-v	Displays verbose information.

3.7.34.F Examples

```
> showstart orion.13762
job orion.13762 requires 2 procs for 0:33:20
Estimated Rsv based start in                 1:04:55 on Fri Jul 15 12:53:40
Estimated Rsv based completion in            2:44:55 on Fri Jul 15 14:33:40
Estimated Priority based start in            5:14:55 on Fri Jul 15 17:03:40
Estimated Priority based completion in       6:54:55 on Fri Jul 15 18:43:40
Estimated Historical based start in         00:00:00 on Fri Jul 15 11:48:45
Estimated Historical based completion in     1:40:00 on Fri Jul 15 13:28:45
Best Partition: fast
```

```
> showstart 12@3600
job 12@3600 requires 12 procs for 1:00:00
Earliest start in         00:01:39 on Wed Aug 31 16:30:45
Earliest completion in     1:01:39 on Wed Aug 31 17:30:45
Best Partition: 32Bit
```

You cannot specify job flags when running *showstart*, and since a job by default can only run on one partition, *showstart* fails when querying for a job requiring more nodes than the largest partition available.

3.7.34.G Additional Information

For reservation based estimates, the information provided by this command is more highly accurate if the job is highest priority, if the job has a reservation, or if the majority of the jobs that are of higher priority have reservations. Consequently, sites wanting to make decisions based on this information might want to consider using the RESERVATIONDEPTH parameter to increase the number of priority based reservations. This can be set so that most or even all idle jobs receive priority reservations and make the results of this command generally useful. The only caution of this approach is that increasing the `RESERVATIONDEPTH` parameter more tightly constrains the decisions of the scheduler and might result in slightly lower system utilization (typically less than 8% reduction).

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.1 checkjob
- 3.7.33 showres
- showstats -f - ESTSTARTTIME (eststarttime)
- showstats -f - AVGQTIME (avgqtime)
- 13.4 Job Start Time Estimates

3.7.35 showstate

3.7.35.A Synopsis

```
showstate [--about] [--help] [--host=<serverHostName>] [--
loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>]
[--timeout=<seconds>] [--version] [--xml]
```

3.7.35.B Overview

This command provides a summary of the state of the system. It displays a list of all active jobs and a text-based map of the status of all nodes and the jobs they are servicing. Basic diagnostic tests are also performed and any problems found are reported.

3.7.35.C Access

By default, this command can be run by any Moab Admin.

3.7.35.D Example

```
> showstate
cluster state summary for Wed Nov 23 12:00:21
    JobID              S       User    Group Procs   Remaining            StartTime
    ------------------ - --------- -------- ----- -----------  -------------------
(A)      fr17n11.942.0 R     johns    staff    16    13:21:15      Nov 22 12:00:21
(B)      fr17n12.942.0 S     johns    staff    32    13:07:11      Nov 22 12:00:21
(C)      fr17n13.942.0 R     johns    staff     8    11:22:25      Nov 22 12:00:21
(D)      fr17n14.942.0 S     johns    staff     8    10:43:43      Nov 22 12:01:21
(E)      fr17n15.942.0 S     johns    staff     8     9:19:25      Nov 22 12:01:21
(F)      fr17n16.942.0 R     johns    staff     8     9:01:16      Nov 22 12:01:21
(G)      fr17n17.942.0 R     johns    staff     1     7:28:25      Nov 22 12:03:22
(H)      fr17n18.942.0 R     johns    staff     1     3:05:17      Nov 22 12:04:22
(I)      fr17n19.942.0 S     johns    staff    24     0:54:38      Nov 22 12:00:22
Usage Summary:  9 Active Jobs  106 Active Nodes
              [0][0][0][0][0][0][0][0][0][1][1][1][1][1][1][1]
              [1][2][3][4][5][6][7][8][9][0][1][2][3][4][5][6]
Frame      2: XXXXXXXXXXXXXXXXXXXXXXXX[ ][A][C][ ][A][C][C][A]
Frame      3: [ ][ ][ ][ ][ ][ ][A][ ][I][ ][I][ ][ ][ ][ ][ ]
Frame      4: [ ][I][ ][ ][ ][A][ ][I][ ][ ][ ][E][ ][I][ ][E]
Frame      5: [F][ ][E][ ][ ][ ][F][F][F][I][ ][ ][E][ ][E][E]
Frame      6: [ ][I][I][E][I][ ][I][I][ ][I][F][I][I][I][I][F]
Frame      7: [ ]XXX[ ]XXX[ ]XXX[ ]XXX[b]XXX[ ]XXX[ ]XXX[#]XXX
Frame      9: [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][E][ ]
Frame     11: [ ][ ][ ][ ][ ][ ][I][F][@][ ][A][I][ ][F][ ][A]
Frame     12: [A][ ][ ][A][ ][ ][C][A][ ][C][A][A][ ][ ][ ][ ]
Frame     13: [D]XXX[I]XXX[ ]XXX[ ]XXX[ ]XXX[ ]XXX[I]XXX[I]XXX
Frame     14: [D]XXX[I]XXX[I]XXX[D]XXX[ ]XXX[H]XXX[I]XXX[ ]XXX
Frame     15: [b]XXX[b]XXX[b]XXX[b]XXX[D]XXX[b]XXX[b]XXX[b]XXX
Frame     16: [b]XXX[ ]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[ ]XXX[b]XXX
Frame     17: [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
Frame     21: [ ]XXX[b]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[b]XXX[b]XXX
Frame     22: [b]XXX[b]XXX[b]XXX[ ]XXX[b]XXX[ ]XXX[b]XXX[b]XXX
Frame     27: [b]XXX[b]XXX[ ]XXX[b]XXX[b]XXX[b]XXX[b]XXX[b]XXX
Frame     28: [G]XXX[ ]XXX[D]XXX[ ]XXX[D]XXX[D]XXX[D]XXX[ ]XXX
Frame     29: [A][C][A][A][C][ ][A][C]XXXXXXXXXXXXXXXXXXXXXXXX
Key:  XXX:Unknown [*]:Down w/Job [#]:Down [']:Idle w/Job [ ]:Idle [@]:Busy w/No Job
[!]:Drained
Key:  [a]:(Any lower case letter indicates an idle node that is assigned to a job)

Check Memory on Node fr3n07
Check Memory on Node fr4n06
Check Memory on Node fr4n09
```

In this example, nine active jobs are running on the system. Each job listed in the top of the output is associated with a letter. For example, job `fr17n11.942.0` is associated with the letter A. This letter can now be used to determine where the job is currently running. By looking at the system map, it can be found that job `fr17n11.942.0` (job A) is running on nodes `fr2n10`, `fr2n13`, `fr2n16`, `fr3n07` ...

The key at the bottom of the system map can be used to determine unusual node states. For example, `fr7n15` is currently in the state down.

After the key, a series of warning messages may be displayed indicating possible system problems. In this case, warning message indicate that there are memory problems on three nodes, `fr3n07`, `fr4n06`, and `fr4n09`. Also, warning messages indicate that job `fr15n09.1097.0` is having difficulty starting. Node `fr11n08` is in state BUSY but has no job assigned to it (it possibly has a runaway job running on it).

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 10.2.2 Racks - specifying node rack / slot location

3.7.36 showstats

3.7.36.A Synopsis

`showstats`

showstats -a [*accountid*] [-v] [-t *<TIMESPEC>*]

showstats -c [*classid*] [-v] [-t *<TIMESPEC>*]

showstats -f *<statistictype>*

showstats -g [*groupid*] [-v] [-t *<TIMESPEC>*]

showstats -j [*jobtemplate*] [-t *<TIMESPEC>*]

showstats -n [*nodeid*] [-t *<TIMESPEC>*]

showstats -q [*qosid*] [-v] [-t *<TIMESPEC>*]

showstats -s

showstats -T [*leafid* | *tree-level*]

showstats -u [*userid*] [-v] [-t *<TIMESPEC>*]

```
[--about] [--help] [--host=<serverHostName>] [--
loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>]
[--timeout=<seconds>] [--version] [--xml]
```

3.7.36.B Overview

This command shows various accounting and resource usage statistics for the system. Historical statistics cover the timeframe from the most recent execution of the mschedctl -f command.

3.7.36.C Access

By default, this command can be run by any Moab level 1, 2, or 3 Admin.

3.7.36.D Options

Option	Description
-a[<ACCOUNTID>]	Display account statistics. See Account Statistics below for an example.
-c[<CLASSID>]	Display class statistics.
-f <statistictype>	Display full matrix statistics (see 3.7.37 showstats -f for more information).
-g[<GROUPID>]	Display group statistics. See Group Statistics below for an example.
-j[<JOBTEMPLATE>]	Display template statistics.
-n[<NODEID>]	Display node statistics (ENABLEPROFILING must be set). See Node Statistics below for an example.
-q [<QOSID>]	Display QoS statistics.
-s	Display general scheduler statistics.

-t	Display statistical information from the specified timeframe:
	`<START_TIME>[,<END_TIME>]`
	`        (ABSTIME: [HH[:MM[:SS]]][_MO[/DD[/YY]]] ie 14:30_06/23)`
	`        (RELTIME: -[[[DD:]HH:]MM:]SS)`
	See Statistics from an Absolute Time Frame and Statistics from a Relative Time Frame below for examples.
	Profiling must be enabled for the credential type you want statistics for. See 2.7.1.E Credential Statistics for information on how to enable profiling. Also, **-t** is not a stand-alone option. It must be used in conjunction with the **-a**, **-c**, **-g**, **-n**, **-q**, or **-u** flag.

-T	Display fairshare tree statistics. See Fairshare Tree Statistics below for an example.
-u[<USERID>]	Display user statistics. See User Statistics below for an example.

Option	Description
-v	Display verbose information. See Verbose Statistics below for an example.

3.7.36.E Examples

Example 3-41: Account Statistics

This example shows a statistical listing of all active accounts. The top line (Account Statistics Initialized...) of the output indicates the beginning of the timeframe covered by the displayed statistics.

```
> showstats -a
Account Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|--------------------------------- Completed ------
----------------------------|
  Account      Jobs Procs ProcHours Jobs    %   PHReq    %    PHDed     %   FSTgt  AvgXF
 MaxXF  AvgQH  Effic  WCAcc
   137651       16    92   1394.52  229  39.15 18486  45.26 7003.5  41.54 40.00   0.77
  8.15   5.21  90.70  34.69
   462212       11    63    855.27   43   7.35  6028  14.76 3448.4  20.45  6.25   0.71
  5.40   3.14  98.64  40.83
   462213        6    72    728.12   90  15.38  5974  14.63 3170.7  18.81  6.25   0.37
  4.88   0.52  82.01  24.14
   005810        3    24    220.72   77  13.16  2537   6.21 1526.6   9.06 -----   1.53
 14.81   0.42  98.73  28.40
   175436        0     0      0.00   12   2.05  6013  14.72  958.6   5.69  2.50   1.78
  8.61   5.60  83.64  17.04
   000102        0     0      0.00    1   0.17    64   0.16    5.1   0.03 -----  10.85
 10.85  10.77  27.90   7.40
   000023        0     0      0.00    1   0.17    12   0.03    0.2   0.00 -----   0.04
  0.04   0.19  21.21   1.20
```

The statistical output is divided into two categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
Account	Account Number.
Jobs	Number of running jobs.
Procs	Number of processors allocated to running jobs.
ProcHours	Number of proc-hours required to complete running jobs.

Column	Description
Jobs*	Number of jobs completed.
%	Percentage of total jobs that were completed by account.
PHReq*	Total proc-hours requested by completed jobs.
%	Percentage of total proc-hours requested by completed jobs that were requested by account.
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total proc-hours dedicated that were dedicated by account.
FSTgt	Fairshare target. An account's fairshare target is specified in the `fs.cfg` file. This value should be compared to the account's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed.
AvgQH*	Average queue time (in hours) of jobs.
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit. A job's wallclock accuracy is capped at 100%, so even if a job exceeds its requested walltime, it will report an accuracy of 100%.

* These fields are empty until an account has completed at least one job.

Example 3-42: Group Statistics

This example shows a statistical listing of all active groups. The top line (Group Statistics Initialized...) of the output indicates the beginning of the timeframe covered by the displayed statistics.

```
> showstats -g
Group Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|--------------------------------- Completed ------
----------------------------|
GroupName  GID Jobs Procs ProcHours Jobs    %   PHReq    %    PHDed    %   FSTgt
AvgXF  MaxXF  AvgQH  Effic  WCAcc
     univ  214   16    92   1394.52  229  39.15 18486  45.26 7003.5  41.54 40.00
0.77   8.15   5.21  90.70  34.69
      daf  204   11    63    855.27   43   7.35  6028  14.76 3448.4  20.45  6.25
0.71   5.40   3.14  98.64  40.83
    dnavy  207    6    72    728.12   90  15.38  5974  14.63 3170.7  18.81  6.25
0.37   4.88   0.52  82.01  24.14
     govt  232    3    24    220.72   77  13.16  2537   6.21 1526.6   9.06 -----
1.53  14.81   0.42  98.73  28.40
      asp  227    0     0      0.00   12   2.05  6013  14.72  958.6   5.69  2.50
1.78   8.61   5.60  83.64  17.04
    derim  229    0     0      0.00   74  12.65   669   1.64  352.5   2.09 -----
0.50   1.93   0.51  96.03  32.60
   dchall  274    0     0      0.00    3   0.51   447   1.10  169.2   1.00 25.00
0.52   0.88   2.49  95.82  33.67
      nih  239    0     0      0.00   17   2.91   170   0.42  148.1   0.88 -----
0.95   1.83   0.14  97.59  84.31
    darmy  205    0     0      0.00   31   5.30   366   0.90   53.9   0.32  6.25
0.14   0.59   0.07  81.33  12.73
  systems   80    0     0      0.00    6   1.03    67   0.16   22.4   0.13 -----
4.07   8.49   1.23  28.68  37.34
      pdc  252    0     0      0.00    1   0.17    64   0.16    5.1   0.03 -----
10.85  10.85  10.77  27.90   7.40
    staff    1    0     0      0.00    1   0.17    12   0.03    0.2   0.00 -----
0.04   0.04   0.19  21.21   1.20
```

The statistical output is divided into two categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
GroupName	Name of group.
GID	Group ID of group.
Jobs	Number of running jobs.
Procs	Number of procs allocated to running jobs.
ProcHours	Number of proc-hours required to complete running jobs.
Jobs*	Number of jobs completed.
%	Percentage of total jobs that were completed by group.

Column	Description
PHReq*	Total proc-hours requested by completed jobs.
%	Percentage of total proc-hours requested by completed jobs that were requested by group.
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total proc-hours dedicated that were dedicated by group.
FSTgt	Fairshare target. A group's fairshare target is specified in the `fs.cfg` file. This value should be compared to the group's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed.
AvgQH*	Average queue time (in hours) of jobs.
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit. A job's wallclock accuracy is capped at 100%, so even if a job exceeds its requested walltime, it will report an accuracy of 100%.

* These fields are empty until a group has completed at least one job.

Example 3-43: Node Statistics

```
> showstats -n
node stats from Mon Jul 10 00:00:00 to Mon Jul 10 16:30:00
node      CfgMem MinMem MaxMem AvgMem | CfgProcs MinLoad MaxLoad AvgLoad
node01     58368      0  21122   5841         32    0.00   32.76   27.62
node02    122880      0  19466    220         30    0.00   33.98   29.54
node03     18432      0   9533   2135         24    0.00   25.10   18.64
node04     60440      0  17531   4468         32    0.00   30.55   24.61
node05     13312      0   2597   1189          8    0.00    9.85    8.45
node06     13312      0   3800   1112          8    0.00    8.66    5.27
```

```
node07      13312      0   2179   1210        8    0.00    9.62    8.27
node08      13312      0   3243   1995        8    0.00   11.71    8.02
node09      13312      0   2287   1943        8    0.00   10.26    7.58
node10      13312      0   2183   1505        8    0.00   13.12    9.28
node11      13312      0   3269   2448        8    0.00    8.93    6.71
node12      13312      0  10114   6900        8    0.00   13.13    8.44
node13      13312      0   2616   2501        8    0.00    9.24    8.21
node14      13312      0   3888    869        8    0.00    8.10    3.85
node15      13312      0   3788    308        8    0.00    8.40    4.67
node16      13312      0   4386   2191        7    0.00   18.37    8.36
node17      13312      0   3158   1870        8    0.00    8.95    5.91
node18      13312      0   5022   2397        8    0.00   19.25    8.19
node19      13312      0   2437   1371        8    0.00    8.98    7.09
node20      13312      0   4474   2486        8    0.00    8.51    7.11
node21      13312      0   4111   2056        8    0.00    8.93    6.68
node22      13312      0   5136   2313        8    0.00    8.61    5.75
node23      13312      0   1850   1752        8    0.00    8.39    5.71
node24      13312      0   3850   2539        8    0.00    8.94    7.80
node25      13312      0   3789   3702        8    0.00   21.22   12.83
node26      13312      0   3809   1653        8    0.00    9.34    4.91
node27      13312      0   5637     70        4    0.00   17.97    2.46
node28      13312      0   3076   2864        8    0.00   22.91   10.33
```

Example 3-44: Verbose Statistics

This example shows a concise summary of the system scheduling state. Note that *`showstats`* and *`showstats -s`* are equivalent.

```
> showstats -v
current scheduler time: Sat Aug 18 18:23:02 2024
moab active for      00:00:01   started on Wed Dec 31 17:00:00
statistics for iteration     0  initialized on Sat Aug 11 23:55:25
Eligible/Idle Jobs:                 6/8      (75.000%)
Active Jobs:                       13
Successful/Completed Jobs:        167/167    (100.000%)
Preempt Jobs:                       0
Avg/Max QTime (Hours):           0.34/2.07
Avg/Max XFactor:                 1.165/3.26
Avg/Max Bypass:                  0.40/8.00
Dedicated/Total ProcHours:      4.46K/4.47K  (99.789%)
Preempt/Dedicated ProcHours:     0.00/4.46K  (0.000%)
Current Active/Total Procs:        32/32     (100.0%)
Current Active/Total Nodes:        16/16      (100.0%)
Avg WallClock Accuracy:          64.919%
Avg Job Proc Efficiency:         99.683%
Min System Utilization:          87.323% (on iteration 46)
Est/Avg Backlog:               02:14:06/03:02:567
```

The first line of output indicates the number of scheduling iterations performed by the current scheduling process, followed by the time the scheduler started. The second line indicates the amount of time the Moab Scheduler has been scheduling in HH:MM:SS notation followed by the statistics initialization time.

The fields are as follows:

Column	Description
Active Jobs	Number of jobs currently active (Running or Starting).
Eligible Jobs	Number of jobs in the system queue (jobs that are considered when scheduling).
Idle Jobs	Number of jobs both in and out of the system queue that are in the LoadLeveler Idle state.
Completed Jobs	Number of jobs completed since statistics were initialized.
Successful Jobs	Jobs that completed successfully without abnormal termination.
XFactor	Average expansion factor of all completed jobs.
Max XFactor	Maximum expansion factor of completed jobs.
Max Bypass	Maximum bypass of completed jobs.
Available ProcHours	Total proc-hours available to the scheduler.
Dedicated ProcHours	Total proc-hours made available to jobs.
Effic	Scheduling efficiency (DedicatedProcHours / Available ProcHours).
Min Efficiency	Minimum scheduling efficiency obtained since scheduler was started.
Iteration	Iteration on which the minimum scheduling efficiency occurred.
Available Procs	Number of procs currently available.
Busy Procs	Number of procs currently busy.
Effic	Current system efficiency (BusyProcs / AvailableProcs).
WallClock Accuracy	Average wallclock accuracy of completed jobs (job-weighted average).
Job Efficiency	Average job efficiency (UtilizedTime / DedicatedTime).

Column	Description
Est Backlog	Estimated backlog of queued work in hours.
Avg Backlog	Average backlog of queued work in hours.

Example 3-45: User Statistics

This example shows a statistical listing of all active users. The top line (User Statistics Initialized...) of the output indicates the timeframe covered by the displayed statistics.

```
> showstats -u
User Statistics Initialized Tue Aug 26 14:32:39
              |----- Running ------|--------------------------------- Completed ------
----------------------------|
 UserName  UID Jobs Procs ProcHours Jobs    %   PHReq    %    PHDed    %   FSTgt
AvgXF  MaxXF  AvgQH  Effic  WCAcc
  moorejt 2617    1    16     58.80    2   0.34   221   0.54 1896.6  11.25 -----
1.02   1.04   0.14  99.52 100.00
    zhong 1767    3    24    220.72   20   3.42  2306   5.65 1511.3   8.96 -----
0.71   0.96   0.49  99.37  67.48
      lui 2467    0     0      0.00   16   2.74  1970   4.82 1505.1   8.93 -----
1.02   6.33   0.25  98.96  57.72
    evans 3092    0     0      0.00   62  10.60  4960  12.14 1464.3   8.69   5.0
0.62   1.64   5.04  87.64  30.62
   wengel 2430    2    64    824.90    1   0.17   767   1.88  630.3   3.74 -----
0.18   0.18   4.26  99.63   0.40
    mukho 2961    2    16     71.06    6   1.03   776   1.90  563.5   3.34 -----
0.31   0.82   0.20  93.15  30.28
  jimenez 1449    1    16    302.29    2   0.34   768   1.88  458.3   2.72 -----
0.80   0.98   2.31  97.99  70.30
     neff 3194    0     0      0.00   74  12.65   669   1.64  352.5   2.09  10.0
0.50   1.93   0.51  96.03  32.60
   cholik 1303    0     0      0.00    2   0.34   552   1.35  281.9   1.67 -----
1.72   3.07  25.35  99.69  66.70
 jshoemak 2508    1    24    572.22    1   0.17   576   1.41  229.1   1.36 -----
0.55   0.55   3.74  99.20  39.20
     kudo 2324    1     8    163.35    6   1.03  1152   2.82  211.1   1.25 -----
0.12   0.34   1.54  96.77   5.67
   xztang 1835    1     8     18.99 ---- ------ ----- ------  176.3   1.05  10.0 -----
- ------ ------  99.62 ------
   feller 1880    0     0      0.00   17   2.91   170   0.42  148.1   0.88 -----
0.95   1.83   0.14  97.59  84.31
    maxia 2936    0     0      0.00    1   0.17   191   0.47  129.1   0.77   7.5
0.88   0.88   4.49  99.84  69.10
 ktgnov71 2838    0     0      0.00    1   0.17   192   0.47   95.5   0.57 -----
0.53   0.53   0.34  90.07  51.20
```

The statistical output is divided into two statistics categories, Running and Completed. Running statistics include information about jobs that are currently running. Completed statistics are compiled using historical information from both running and completed jobs.

The fields are as follows:

Column	Description
UserName	Name of user.
UID	User ID of user.
Jobs	Number of running jobs.
Procs	Number of procs allocated to running jobs.
ProcHours	Number of proc-hours required to complete running jobs.
Jobs*	Number of jobs completed.
%	Percentage of total jobs that were completed by user.
PHReq*	Total proc-hours requested by completed jobs.
%	Percentage of total proc-hours requested by completed jobs that were requested by user.
PHDed	Total proc-hours dedicated to active and completed jobs. The proc-hours dedicated to a job are calculated by multiplying the number of allocated procs by the length of time the procs were allocated, regardless of the job's CPU usage.
%	Percentage of total proc-hours dedicated that were dedicated by user.
FSTgt	Fairshare target. A user's fairshare target is specified in the `fs.cfg` file. This value should be compared to the user's node-hour dedicated percentage to determine if the target is being met.
AvgXF*	Average expansion factor for jobs completed. A job's XFactor (expansion factor) is calculated by the following formula: (QueuedTime + RunTime) / WallClockLimit.
MaxXF*	Highest expansion factor received by jobs completed.
AvgQH*	Average queue time (in hours) of jobs.
Effic	Average job efficiency. Job efficiency is calculated by dividing the actual node-hours of CPU time used by the job by the node-hours allocated to the job.
WCAcc*	Average wallclock accuracy for jobs completed. Wallclock accuracy is calculated by dividing a job's actual run time by its specified wallclock limit.

Column	Description
	A job's wallclock accuracy is capped at 100%, so even if a job exceeds its requested walltime, it will report an accuracy of 100%.

* These fields are empty until a user has completed at least one job.

Example 3-46: Fairshare Tree Statistics

```
> showstats -T
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|---------------------------------- Completed
-----------------------------------|
user           Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
root              0     0      0.00    0   56 100.00  2.47K 100.00  1.58K  48.87 -----
   1.22   0.00   0.24 100.00  58.84
 l1.1             0     0      0.00    0   25  44.64 845.77  34.31 730.25  22.54 -----
   1.97   0.00   0.20 100.00  65.50
  Administrati    0     0      0.00    0   10  17.86 433.57  17.59 197.17   6.09 -----
   3.67   0.00   0.25 100.00  62.74
  Engineering     0     0      0.00    0   15  26.79 412.20  16.72 533.08  16.45 -----
   0.83   0.00   0.17 100.00  67.35
 l1.2             0     0      0.00    0   31  55.36  1.62K  65.69 853.00  26.33 -----
   0.62   0.00   0.27 100.00  53.46
  Shared          0     0      0.00    0    3   5.36  97.17   3.94  44.92   1.39 -----
   0.58   0.00   0.56 100.00  31.73
  Test            0     0      0.00    0    3   5.36  14.44   0.59  14.58   0.45 -----
   0.43   0.00   0.17 100.00  30.57
  Research        0     0      0.00    0   25  44.64  1.51K  61.16 793.50  24.49 -----
   0.65   0.00   0.24 100.00  58.82

> showstats -T 2
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|---------------------------------- Completed
-----------------------------------|
user           Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
  Test            0     0      0.00    0   22   4.99 271.27   0.55 167.42   0.19 -----
   3.86   0.00   2.89 100.00  60.76
  Shared          0     0      0.00    0   59  13.38 12.30K  24.75  4.46K   5.16 -----
   6.24   0.00  10.73 100.00  49.87
  Research        0     0      0.00    0  140  31.75  9.54K  19.19  5.40K   6.25 -----
   2.84   0.00   5.52 100.00  57.86
  Administrati    0     0      0.00    0   84  19.05  7.94K  15.96  4.24K   4.91 -----
   4.77   0.00   0.34 100.00  62.31
  Engineering     0     0      0.00    0  136  30.84 19.67K  39.56 28.77K  33.27 -----
   3.01   0.00   3.66 100.00  63.70

> showstats -T l1.1
statistics initialized Mon Jul 10 15:29:41
              |-------- Active ---------|---------------------------------- Completed
-----------------------------------|
user           Jobs Procs ProcHours  Mem Jobs    %    PHReq    %    PHDed    %   FSTgt
  AvgXF  MaxXF  AvgQH  Effic  WCAcc
 l1.1             0     0      0.00    0  220  49.89 27.60K  55.52 33.01K  38.17 -----
   3.68   0.00   2.39 100.00  63.17
  Administrati    0      0       0.00    0   84   19.05  7.94K  15.96  4.24K   4.91 -----
   4.77   0.00   0.34 100.00   62.31
  Engineering     0      0       0.00    0  136   30.84 19.67K  39.56 28.77K  33.27 -----
   3.01   0.00   3.66 100.00   63.70
```

Example 3-47: Statistics from an Absolute Time Frame

Moab returns information about the class `batch` from January 1, 2024 to December 31, 2024. For more information about specifying absolute dates, see Absolute Time Format in TIMESPEC below.

```
> showstats -c batch -v -t 00:00:01_01/01/24,23:59:59_12/31/24
statistics initialized Wed Jan 1 00:00:00

-------- Active ---------       --------------------- Completed ---------------------

class Jobs Procs ProcHours Mem Jobs     %    PHReq      %    PHDed      %     FSTgt AvgXF
MaxXF  AvgQH  Effic  WCAcc
batch     0      0       0.00    0   23  100.00    15  100.00     1  100.00  -----   0.40
5.01   0.00  88.94  39.87
```

Example 3-48: Statistics from a Relative Time Frame

Moab returns information about user `bob` from the past 30 days. For more information about specifying relative dates, see Relative Time Format in TIMESPEC below.

```
> showstats -u bob -v -t -30:00:00:00
statistics initialized Mon Nov 11 15:30:00

-------- Active ---------       --------------------- Completed ---------------------

user Jobs Procs ProcHours Mem Jobs     %    PHReq      %    PHDed      %     FSTgt AvgXF
MaxXF  AvgQH  Effic  WCAcc
bob      0      0       0.00    0   23  100.00    15  100.00     1  100.00  -----   0.40
5.01   0.00  88.94  39.87
```

3.7.36.F TIMESPEC

Relative Time Format

The relative time format specifies a time by using the current time as a reference and specifying a time offset.

Format

+[[[DD:]HH:]MM:]SS

Examples

2 days, 3 hours and 57 seconds in the future:

```
+02:03:0:57
```

21 days (3 weeks) in the future:

```
+21:0:0:0
```

30 seconds in the future:

```
+30
```

Absolute Time Format

The absolute time format specifies a specific time in the future.

Format

[HH[:MM[:SS]]][_MO[/DD[/YY]]] (i.e., 14:30_06/23)

Examples

1 PM, March 1 (this year)

```
13:00_03/01
```

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- -f - FLUSH - command that re-initializes statistics
- 3.7.37 showstats -f - command that displays full matrix statistics

3.7.37 showstats -f

3.7.37.A Synopsis

```
showstats -f <statistictype> [--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]
```

3.7.37.B Overview

Shows table of various scheduler statistics. This command displays a table of the selected Moab Scheduler statistics, such as expansion factor, bypass count, jobs, proc-hours, wallclock accuracy, and backfill information.

Statistics are aggregated over time. This means statistical information is not available for time frames and the `-t` option is not supported with *`showstats -f`*.

3.7.37.C Access

This command can be run by any Moab Scheduler Admin.

3.7.37.D Options

Options	Description
AVGBYPASS	The number of times a priority job has been 'bypassed' by backfill, allowing a lower priority job to run ahead of it. See Example 3-50 for more information.
AVGQTIME	Average queue time. Includes summary of job-weighted queue time and total samples.
AVGXFACTOR	Average expansion factor. Includes summary of job-weighted expansion factor, processor-weighted expansion factor, processor-hour-weighted expansion factor, and total number of samples.
BFCOUNT	Number of jobs backfilled. Includes summary of job-weighted backfill job percent and total samples.
BFPHRUN	Number of proc-hours backfilled. Includes summary of job-weighted backfill proc-hour percentage and total samples.
ESTSTARTTIME	Job start time estimate for jobs meeting specified processor/duration criteria. This estimate is based on the reservation start time analysis algorithm.
JOBCOUNT	Number of jobs. Includes summary of total jobs and total samples.
MAXBYPASS	Maximum bypass count. Includes summary of overall maximum bypass and total samples.
MAXXFACTOR	Maximum expansion factor. Includes summary of overall maximum expansion factor and total samples.
PHREQUEST	proc-hours requested. Includes summary of total proc-hours requested and total samples.

Options	Description
PHRUN	proc-hours run. Includes summary of total proc-hours run and total samples.
QOSDELIVERED	Quality of service delivered. Includes summary of job-weighted quality of service success rate and total samples.
WCACCURACY	Wallclock accuracy. Includes summary of overall wall clock accuracy and total samples.

3.7.37.E Examples

Example 3-49: Average XFactor Grid

The *showstats -f* command returns a table with data for the specified STATISTICTYPE parameter:

```
> showstats -f AVGXFACTOR
Average XFactor Grid
[ NODES ][ 00:02:00 ][ 00:04:00 ][ 00:08:00 ][ 00:16:00 ][ 00:32:00 ][ 01:04:00 ][
02:08:00 ][ 04:16:00 ][ 08:32:00 ][ 17:04:00 ][ 34:08:00 ][   TOTAL  ]
[    1  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[    2  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[    4  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][
1.00    1][ -------- ][  1.12    2][ -------- ][ -------- ][  1.10    3]
[    8  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][
1.00    2][  1.24    2][ -------- ][ -------- ][ -------- ][  1.15    4]
[   16  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01    2][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01    2]
[   32  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[   64  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[  128  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[  256  ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][ ---
----- ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ]
[ T TOT ][ -------- ][ -------- ][ -------- ][ -------- ][ -------- ][  1.01    2][
1.00    3][  1.24    2][  1.12    2][ -------- ][ -------- ]
Job Weighted X Factor:     1.0888
Node Weighted X Factor:    1.1147
NS Weighted X Factor:      1.1900
Total Samples:                  9
```

The left-most column shows the maximum number of processors required by the jobs shown in the other columns. The column headers indicate the maximum wallclock time (in HH:MM:SS notation) requested by the jobs shown in the columns. The data returned in the table varies by the STATISTICTYPE requested. For table entries with one number, it is of the data requested. For table entries with two numbers, the left number is the data requested and the right number is the number of jobs used to calculate the average. Table entries that contain only dashes (-------) indicate no job has completed that matches the profile associated for this inquiry. The bottom row shows the totals for each column. Following each table is a summary, which varies by the STATISTICTYPE requested.

The column and row break down can be adjusted using the STATPROC* and STATTIME* parameters respectively.

This particular example shows the average expansion factor grid. Each table entry indicates two pieces of information — the average expansion factor for all jobs that meet this slot's profile and the number of jobs that were used to calculate this average. For example, the XFactors of two jobs were averaged to obtain an average XFactor of 1.24 for jobs requiring over 2 hours 8 minutes, but not more than 4 hours 16 minutes and between 5 and 8 processors. Totals along the bottom provide overall XFactor averages weighted by job, processors, and processor-hours.

Example 3-50: Average Bypass

```
> showstats -f AVGBYPASS
Average Bypass (bypass count)
[ PROCS ][ 0:15:00 ][ 1:00:00 ][ 4:00:00 ][ 16:00:00 ][ 64:00:00 ][ 256:00:00 ][ TOTAL
]
[ 1 ][ 0.00 10][ 0.00 70][ 0.00 31][ 0.00 34][ 0.00 6][------------][ 0.00 150]
[ 4 ][------------][------------][------------][------------][------------][----------
--][------------]
[ 16 ][ 0.08 37687][ 0.08 164307][ 0.32 117767][ 0.10 34073][ 0.58 1282][------------]
[ 0.16 355116]
[ 64 ][ 0.18 769][ 0.13 1839][ 0.18 8084][ 0.82 2812][ 0.00 34][------------][ 0.31
13538]
[ 256 ][ 0.39 316][ 1.40 778][ 4.40 494][ 1.77 28917][ 0.33 6][------------][ 1.79
30511]
[ TOTAL ][ 0.08 38782][ 0.09 166994][ 0.33 126376][ 0.86 65835][ 0.57 1328][----------
--]
Job Weighted X Bypass: 0.2932
Total Samples: 399315
```

The `showstats -f` command returns a table with data for the specified STATISTICTYPE parameter, in this case for AVGBYPASS. In this particular example, the upper left cell indicates that 10 jobs were run by Moab, which had 0-15 minutes of requested walltime and 0-1 procs allocated. The 0.00 indicates that of the 10 jobs, the average number of times the jobs were bypassed was 0, meaning it did not occur. Further, looking at row 256 and column 4:00, we see that 494 jobs have been run by Moab that meet this criteria. On average, these jobs were each bypassed 4.40 times.

Related Topics

- (Optional) Install Moab Client - in the *Moab HPC Suite Installation and Configuration Guide* explains how to distribute this command to client nodes
- 3.7.25 mschedctl (-f - FLUSH command)
- 3.7.36 showstats (command)
- STATPROCMIN parameter
- STATPROCSTEPCOUNT parameter
- STATPROCSTEPSIZE parameter
- STATTIMEMIN parameter
- STATTIMESTEPCOUNT parameter
- STATTIMESTEPSIZE parameter

3.7.38 Deprecated Commands

In this topic:

3.7.38.A canceljob

This command is deprecated. Use mjobctl -c instead.

Synopsis

`canceljob` *jobid* [*jobid*]...

Overview

The `canceljob` command is used to selectively cancel the specified job(s) (active, idle, or non-queued) from the queue.

Access

This command can be run by any Moab Admin and by the owner of the job (see the parameter ADMINCFG).

Flag	Name	Format	Default	Description	Example
-h	HELP		N/A	Display usage information.	`> canceljob -h`
	JOB ID	<STRING>	---	A jobid, a job expression, or the keyword ALL.	`> canceljob 13001 13003`

Example

Example 3-51: Cancel job 6397

```
> canceljob 6397
```

3.7.38.B changeparam

This command is deprecated. Use mschedctl -m instead.

Synopsis

`changeparam` *parameter value*

Overview

The `changeparam` command is used to dynamically change the value of any parameter that can be specified in the `moab.cfg` file. The changes take effect at the beginning of the next scheduling iteration. They are not persistent, only lasting until Moab is shut down.

`changeparam` is a compact command of mschedctl -m.

Access

This command can be run by a level 1 Moab Admin.

3.7.38.C diagnose

This command is deprecated. Use mdiag instead.

Synopsis

diagnose -a [*accountid*]

diagnose -b [-l *policylevel*] [-t *partition*]

diagnose -c [*classid*]

diagnose -C [*configfile*]

diagnose -f [-o user|group|account|qos|class]

diagnose -g [*groupid*]

diagnose -j [*jobid*]

diagnose -L

diagnose -m [*rackid*]

diagnose -n [-t *partition*] [*nodeid*]

diagnose -p [-t *partition*]

diagnose -q [*qosid*]

diagnose -r [*reservationid*]

diagnose -R [*resourcemanagername*]

diagnose -s [*standingreservationid*]

diagnose -S diagnose -u [*userid*]

diagnose -v

diagnose -x

Overview

The `diagnose` command is used to display information about various aspects of scheduling and the results of internal diagnostic tests.

3.7.38.D releasehold

This command is deprecated. Use mjobctl -u instead.

Synopsis

releasehold [-a|-b] jobexp

Overview

Release hold on specified job(s).

This command enables you to release batch holds or all holds (system, user, and batch) on specified jobs. Any number of jobs can be released with this command.

Access

By default, this command can be run by any Moab Scheduler Admin.

Parameters

JOBEXP	Job expression of job(s) to release.

Flags

-a	Release all types of holds (user, system, batch) for specified job(s).
-b	Release batch hold from specified job(s).
-h	Help for this command.

Examples

Example 3-52: releasehold -b

A batch hold was released from this one job:

```
> releasehold -b 6443
batch hold released for job 6443
```

Example 3-53: releasehold -a

All holds were released from the specified jobs:

```
> releasehold -a "81[1-6]"
holds modified for job 811
holds modified for job 812
holds modified for job 813
holds modified for job 814
holds modified for job 815
holds modified for job 816
```

Related Topics

- 3.7.38.H sethold
- 3.7.20 mjobctl

3.7.38.E releaseres

This command is deprecated. Use mrsvctl -r instead.

Synopsis

`releaseres [arguments] reservationid [reservationid...]`

Overview

Release existing reservation.

This command allows Moab Scheduler Admins to release any user, group, account, job, or system reservation. Users are allowed to release reservations on jobs they own. Note that releasing a reservation on an active job has no effect since the reservation will be automatically recreated.

Access

Users can use this command to release any reservation they own. Level 1 and level 2 Moab Admins can use this command to release any reservation.

Parameters

RESERVATION ID	Name of reservation to release.

Example

Example 3-54: Release two existing reservations

```
> releaseres system.1 bob.2
released User reservation 'system.1'
released User reservation 'bob.2'
```

3.7.38.F resetstats

This command is deprecated. Use mschedctl -f instead.

Synopsis

`resetstats`

Overview

This command resets all internally-stored Moab Scheduler statistics to the initial start-up state as of the time the command was executed.

Access

By default, this command can be run by level 1 scheduler admins.

Example

```
> resetstats Statistics Reset at time Wed Feb 25 23:24:55 2025
```

3.7.38.G runjob

This command is deprecated. Use mjobctl -x instead.

Synopsis

`runjob` [-c|-f|-n *nodelist*|-p *partition*|-s|-x] *jobid*

Overview

This command will attempt to immediately start the specified job.

`runjob` is a deprecated command, replaced by mjobctl.

Access

By default, this command can be run by any Moab Admin.

Parameters

JOBID	Name of the job to run.

Args	Description
-c	*Clear* job parameters from previous runs (used to clear PBS neednodes attribute after PBS job launch failure).
-f	Attempt to *force* the job to run, ignoring throttling policies.
-n <NODELIST>	Attempt to start the job using the specified *nodelist* where nodenames are comma- or colon-delimited.
-p <PARTITION>	Attempt to start the job in the specified *partition*.
-s	Attempt to *suspend* the job.
-x	Attempt to force the job to run, ignoring throttling policies, QoS constraints, and reservations.

Example

Example 3-55: Run job cluster.231

```
> runjob cluster.231
job cluster.231 successfully started
```

See Also

- 3.7.20 mjobctl
- 3.7.38.A canceljob - cancel a job
- 3.7.1 checkjob - show detailed status of a job
- 3.7.31 showq - list queued jobs

3.7.38.H sethold

This command is deprecated. Use mjobctl -h instead.

Synopsis

`sethold` [-b] *jobid* [*jobid...*]

Overview

Set hold on specified job(s).

Permissions

This command can be run by any Moab Scheduler Admin.

Parameters

JOB	Job number of job to hold.

Flags

-b	Set a batch hold. Typically, only the scheduler places batch holds. This flag allows an admin to manually set a batch hold.
-h	Help for this command.

Example

```
> sethold -b fr17n02.1072.0 fr15n03.1017.0
Batch Hold Placed on All Specified Jobs
```

In this example, a batch hold is placed on job `fr17n02.1072.0` and job `fr15n03.1017.0`.

3.7.38.I setqos

This command is deprecated. Use mjobctl -m instead.

Synopsis

`setqos` `qosid jobid`

Overview

Set Quality Of Service for a specified job. This command allows users to change the QOS of their own jobs.

Access

This command can be run by any user.

Parameters

JOBID	Job name.
QOSID	QOS name.

Example

```
> setqos high_priority moab.3

Job QOS Adjusted
```

This example sets the Quality Of Service to a value of `high_priority` for job `moab.3`.

3.7.38.J setres

This command is deprecated. Use mrsvctl -c instead.

Synopsis

`setres` `[arguments] resourceexpression`
[-a <ACCOUNT_LIST>]
[-b <SUBTYPE>]
[-c <CHARGE_SPEC>]
[-d <DURATION>]
[-e <ENDTIME>]
[-E] // EXCLUSIVE
[-f <FEATURE_LIST>]
[-g <GROUP_LIST>]
[-n <NAME>]
[-o <OWNER>]
[-p <PARTITION>]
[-q <QUEUE_LIST>] // (i.e., CLASS_LIST)
[-Q <QOSLIST>]
[-r <RESOURCE_DESCRIPTION>]
[-R <RESERVATION_PROFILE>]
[-s <STARTTIME>]
[-T <TRIGGER>]
[-u <USER_LIST>]
[-x <FLAGS>]

Overview

Reserve resources for use by jobs with particular credentials or attributes.

Access

This command can be run by level 1 and level 2 Moab Admins.

Parameters

Name	Format	Default	Description
ACCOUNT_LIST	`<STRING>`[:`<STRING>`]...	---	List of accounts that will be allowed access to the reserved resources.
SUBTYPE	`<STRING>`	---	Specify the subtype for a reservation.
CHARGE_SPEC	`<ACCOUNT>`[,`<GROUP>`[,`<USER>`]]	---	Specifies which credentials will be accountable for unused resources dedicated to the reservation.
CLASS_LIST	`<STRING>`[:`<STRING>`]...	---	List of classes that will be allowed access to the reserved resource.
DURATION	[[[DD:]HH:]MM:]SS	INFINITY	Duration of the reservation (not needed if ENDTIME is specified).
ENDTIME	[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS	INFINITY	Absolute or relative time reservation will end (not required if Duration specified).
EXCLUSIVE	N/A	N/A	Requests exclusive access to resources.
FEATURE_LIST	`<STRING>`[:`<STRING>`]...	---	List of node features that must be possessed by the reserved resources.
FLAGS	`<STRING>`[:`<STRING>`]...	---	List of reservation flags (see 6.1.5 Configuring and Managing Reservations for details).
GROUP_LIST	`<STRING>`[:`<STRING>`]...	---	List of groups that will be allowed access to the reserved resources.
NAME	`<STRING>`	Name set to first name listed in ACL or `SYSTEM` if no ACL specified	Name for new reservation.

Name	Format	Default	Description
OWNER	`<CREDTYPE>:<CREDID>` where `CREDTYPE` is one of user, group, acct, class, or qos	N/A	Specifies which credential is granted reservation ownership privileges.
PARTITION	`<STRING>`	[ANY]	Partition where resources must be located.
QOS_LIST	`<STRING>`[:`<STRING>`]...	---	List of QOSs that will be allowed access to the reserved resource.
RESERVATION_ PROFILE	Existing reservation profile ID	N/A	Requests that default reservation attributes be loaded from the specified reservation profile (see the parameter RSVPROFILE).
RESOURCE_ DESCRIPTION	Colon-delimited list of zero or more of the following `<ATTR>=<VALUE>` pairs PROCS=`<INTEGER>` MEM=`<INTEGER>` DISK=`<INTEGER>` SWAP=`<INTEGER>` GRES=`<STRING>`	PROCS=-1	The resources to be reserved per task. (-1 indicates all resources on node).
RESOURCE_ EXPRESSION	ALL or TASKS {`==` | `>=`}`<TASKCOUNT>` or `<HOST_REGEX>`	Required Field. No Default	The tasks to reserve. ALL indicates all resources available should be reserved. If ALL or a host expression is specified, Moab will apply the reservation regardless of existing reservations and exclusive issues. If TASKS is used, Moab will only allocate accessible resources.

Name	Format	Default	Description
STARTTIME	[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS	NOW	Absolute or relative time reservation will start.
TRIGGER	<STRING>	N/A	Comma-delimited reservation trigger list following format described in the trigger format section of the reservation configuration overview.
USER_LIST	<STRING>[:<STRING>]...	---	List of users that will be allowed access to the reserved resources.

Description

The *setres* command allows an arbitrary block of resources to be reserved for use by jobs that meet the specified access constraints. The timeframe covered by the reservation can be specified on either an absolute or relative basis. Only jobs with credentials listed in the reservation ACL (i.e., `USERLIST`, `GROUPLIST`,...) can utilize the reserved resources. However, these jobs still have the freedom to utilize resources outside of the reservation. The reservation will be assigned a name derived from the ACL specified. If no reservation ACL is specified, the reservation is created as a system reservation and no jobs will be allowed access to the resources during the specified timeframe (valuable for system maintenance, etc.). See 6.1.1 Reservation Overview for more information.

Reservations can be viewed using the showres command and can be released using the releaseres command.

Examples

Example 3-56: setres -u

Reserve two nodes for use by users `john` and `mary` for a period of 8 hours starting in 24 hours:

```
> setres -u john:mary -s +24:00:00 -d 8:00:00 TASKS==2
reservation 'john.1' created on 2 nodes (2 tasks)
node001:1
node005:1
```

Example 3-57: setres -s

Schedule a system wide reservation to allow system maintenance on Jun 23, 8:00 A.M. until Jun 23, 5:00 P.M.:

```
> setres -s 8:00:00_06/23 -e 17:00:00_06/23 ALL
reservation 'system.1' created on 8 nodes (8 tasks)
node001:1
node002:1
node003:1
node004:1
node005:1
node006:1
node007:1
node008:1
```

Example 3-58: setres -r

Reserve 1 processor and `512` MB of memory on nodes `node003` through node `node006` for members of the group `staff` and jobs in the `interactive` class:

```
> setres -r PROCS=1:MEM=512 -g staff -l interactive 'node00[3-6]'
reservation 'staff.1' created on 4 nodes (4 tasks)
node003:1
node004:1
node005:1
node006:1
```

3.7.38.K setspri

This command is deprecated. Use mjobctl -p instead.

Synopsis

`setspri` [-r] *priorityjobid*

Overview

This command is deprecated by the mjobctl command.

Set or remove absolute or relative system priorities for a specified job. This command enables you to set or remove a system priority level for a specified job. Any job with a system priority level set is guaranteed a higher priority than jobs without a system priority. Jobs with higher system priority settings have priority over jobs with lower system priority settings.

Access

This command can be run by any Moab Scheduler Admin.

Parameters

JOB	Name of job.

PRIORITY	System priority level. By default, this priority is an absolute priority overriding the policy generated priority value. Range is 0 to clear, 1 for lowest, 1000 for highest. The given value is added onto the system priority (see 32-bit and 64-bit values below), except for a given value of zero. If the '-r' flag is specified, the system priority is relative, adding or subtracting the specified value from the policy generated priority. If a relative priority is specified, any value in the range +/- 1,000,000,000 is acceptable.

Flags

-r	Set relative system priority on job.

Examples

Example 3-59: setspri 10

A system priority of `10` is set for job `orion.4752`:

```
> setspri 10 orion.4752
job system priority adjusted
```

Example 3-60: setspri 0

System priority is cleared for job `clusterB.1102`:

```
> setspri 0 clusterB.1102
job system priority adjusted
```

Example 3-61: setspri -r

The job's priority will be increased by `100000` over the value determine by configured priority policy:

```
> setspri -r 100000 job.00001
job system priority adjusted
```

This command is deprecated. Use mjobctl instead.

3.7.38.L showconfig

Synopsis

showconfig [-v] [--about] [--help] [--host=<serverHostName>] [--loglevel=<logLevel>] [--msg=<message>] [--port=<serverPort>] [--timeout=<seconds>] [--version] [--xml]

Overview

View the current configurable parameters of the Moab Scheduler. The *showconfig* command shows the current scheduler version and all scheduler parameters. These parameters are set via internal defaults, command line arguments, environment variable settings, parameters in the `moab.cfg` file, and via the mschedctl -m command. Because of the many sources of configuration settings, the output may differ from the contents of the `moab.cfg` file. The output is such that it can be saved and used as the contents of the `moab.cfg` file if desired.

The showconfig command does not show credential parameters (such as user, group class, QoS, account).

Access

This command can be run by a level 1, 2, or 3 Moab Admin.

Flags

-h	Help for this command.
-v	This optional flag turns on verbose mode, which shows all possible Moab Scheduler parameters and their current settings. If this flag is not used, this command operates in context-sensitive terse mode, which shows only certain parameter settings.

Example

```
> showconfig
# moab scheduler version x.x.x (PID: 11080)
BACKFILLPOLICY                  FIRSTFIT
BACKFILLMETRIC                  NODES
ALLOCATIONPOLICY                MINRESOURCE
RESERVATIONPOLICY               CURRENTHIGHEST
...
```

The *showconfig* command without the `-v` flag does not show the settings of all scheduling parameters. To show the settings of all scheduling parameters, use the `-v` (verbose) flag. This will provide an extended output. This output is often best used in conjunction with the *grep* command as the output can be voluminous.

Related Topics

- 3.7.25 mschedctl (command that changes the various Moab Scheduler parameters)
- See Appendix A: Moab Parameters for details about configurable parameters

Chapter 4: Prioritizing Jobs and Allocating Resources

In this chapter:

4.1 Job Prioritization

In general, prioritization is the process of determining which of many options best fulfills overall goals. In the case of scheduling, a site will often have multiple, independent goals that may include maximizing system utilization, giving preference to users in specific projects, or making certain that no job sits in the queue for more than a given period of time. The approach used by Moab in representing a multi-faceted set of site goals is to assign weights to the various objectives so an overall value or priority can be associated with each potential scheduling decision. With the jobs prioritized, the scheduler can roughly fulfill site objectives by starting the jobs in priority order.

In this section:

Related Topics

- 3.7.11 mdiag -p (priority diagnostics)

4.1.1 Job Priority Overview

Moab's prioritization mechanism allows component and subcomponent weights to be associated with many aspects of a job to enable fine-grained control over this aspect of scheduling. To allow this level of control, Moab uses a simple priority-weighting hierarchy where the contribution of each priority subcomponent is calculated as follows:

<COMPONENT WEIGHT> * <SUBCOMPONENT WEIGHT> * <PRIORITY SUBCOMPONENT VALUE>

Each priority component contains one or more subcomponents as described in the section Job Priority Factors. For example, the Resource component consists of Node, Processor, Memory, Swap, Disk, Walltime, and PE subcomponents. While there are numerous priority components and many more subcomponents, you need only focus on and configure the subset of components related to your particular priority needs. In actual usage, few sites use more than a small fraction (usually 5 or fewer) of the available priority subcomponents. This results in fairly straightforward priority configurations and tuning. By mixing and matching priority weights, sites can generally obtain the desired job-start behavior. At any time, you can issue the mdiag -p command to determine the impact of the current priority-weight settings on idle jobs. Likewise, the command showstats -f can assist the admin in evaluating priority effectiveness on historical system usage metrics such as queue time or expansion factor.

As mentioned above, a job's priority is the weighted sum of its activated subcomponents. By default, the value of all component and subcomponent weights is set to 1 and 0 respectively. The one exception is the 'QUEUETIME' subcomponent weight that is set to 1. This results in a total job priority equal to the period of time the job has been queued, causing Moab to act as a simple FIFO. Once the summed component weight is determined, this value is then bounded resulting in a priority ranging between 0 and MAX_PRIO_VAL, which is currently defined as 1000000000 (one billion). In no case will a job obtain a priority in excess of MAX_PRIO_VAL through its priority subcomponent values.

Negative priority jobs can be allowed if desired; see the parameters ENABLENEGJOBPRIORITY and REJECTNEGPRIOJOBS for more information.

Using the mjobctl -p command, site admins can adjust the base calculated job priority by either assigning a relative priority adjustment or an absolute system priority. A relative priority adjustment causes the base priority to be increased or decreased by a specified value. Setting an absolute system priority, SPRIO, causes the job to receive a priority equal to MAX_PRIO_VAL + SPRIO, and therefore guaranteed to be of higher value than any naturally occurring job priority.

Related Topics

- REJECTNEGPRIOJOBS parameter

4.1.2 Job Priority Factors

In this topic:

4.1.2.A Job Priority Factors and Subfactors

Moab allows jobs to be prioritized based on a range of job related factors. These factors are broken down into a two-tier hierarchy of priority factors and subfactors, each of which can be independently assigned a weight. This approach provides the admin with detailed yet straightforward control of the job selection process.

Each factor and subfactor can be configured with independent priority weight and priority cap values (described later). In addition, per credential and per QoS priority weight adjustments can be specified for a subset of the priority factors. For example, QoS credentials can adjust the queuetime subfactor weight and group credentials can adjust fairshare subfactor weight.

The following table highlights the factors and subfactors that make up a job's total priority:

Factor	SubFactor	Metric
CRED **(job credentials)**	USER	user-specific priority (see the parameter USERCFG)
	GROUP	group-specific priority (see the parameter GROUPCFG)

Factor	SubFactor	Metric
	ACCOUNT	account-specific priority (see the parameter ACCOUNTCFG)
	QOS	QoS-specific priority (see the parameter QOSCFG)
	CLASS	class/queue-specific priority (see the parameter CLASSCFG)
FS (fairshare usage)	FSUSER	user-based historical usage (see 5.3 Fairshare)
	FSGROUP	group-based historical usage (see 5.3 Fairshare)
	FSACCOUNT	account-based historical usage (see 5.3 Fairshare)
	FSQOS	QoS-based historical usage (see 5.3 Fairshare)
	FSCLASS	class/queue-based historical usage (see 5.3 Fairshare)
	FSGUSER	imported global user-based historical usage (see 18.4 Identity Managers and 5.3 Fairshare)
	FSGGROUP	imported global group-based historical usage (see 18.4 Identity Managers and 5.3 Fairshare)
	FSGACCOUNT	imported global account-based historical usage (see 18.4 Identity Managers and 5.3 Fairshare)
	FSJPU	current active jobs associated with job user
	FSPPU	current number of processors allocated to active jobs associated with job user
	FSPSPU	current number of processor-seconds allocated to active jobs associated with job user
	WCACCURACY	user's current historical job wallclock accuracy calculated as total processor-seconds dedicated / total processor-seconds requested

Factor	SubFactor	Metric
		Factor values are in the range of 0.0 to 1.0.
RES **(requested job resources)**	NODE	number of nodes requested
	PROC	number of processors requested
	MEM	total real memory requested (in MB)
	SWAP	total virtual memory requested (in MB)
	DISK	total local disk requested (in MB)
	PS	total processor-seconds requested
	PE	total processor-equivalent requested
	WALLTIME	total walltime requested (in seconds)
SERV **(current service levels)**	QUEUETIME	time job has been queued (in minutes)
	XFACTOR	minimum job expansion factor
	BYPASS	number of times job has been bypassed by backfill
	STARTCOUNT	number of times job has been restarted
	DEADLINE	proximity to job deadline
	SPVIOLATION	Boolean indicating whether the active job violates a soft usage limit
	USERPRIO	user-specified job priority
TARGET **(target service levels)**	TARGETQUEUETIME	time until queuetime target is reached (exponential)
	TARGETXFACTOR	distance to target expansion factor (exponential)

Factor	SubFactor	Metric
USAGE (consumed resources -- active jobs only)	CONSUMED	processor-seconds dedicated to date
	REMAINING	processor-seconds outstanding
	PERCENT	percent of required walltime consumed
	EXECUTIONTIME	seconds since job started
ATTR (job attribute-based prioritization)	ATTRATTR	Attribute priority if specified job attribute is set (attributes can be user-defined or one of **preemptor**, or **preemptee**). Default is 0.
	ATTRSTATE	Attribute priority if job is in specified state (see 2.3.1.A Job States). Default is 0.
	ATTRGRES	Attribute priority if a generic resource is requested. Default is 0.

*CAP parameters (FSCAP, for example) are available to limit the maximum absolute value of each priority component and subcomponent. If set to a positive value, a priority cap will bound priority component values in both the positive and negative directions.

All *CAP and *WEIGHT parameters are specified as positive or negative integers. Non-integer values are not supported.

4.1.2.B Credential (CRED) Component

The credential component enables you to prioritize jobs based on political issues such as the relative importance of certain groups or accounts. This allows direct political priorities to be applied to jobs.

The priority calculation for the credential component is as follows:

Priority += CREDWEIGHT * (
USERWEIGHT * Job.User.Priority +
GROUPWEIGHT * Job.Group.Priority +
ACCOUNTWEIGHT * Job.Account.Priority +
QOSWEIGHT * Job.Qos.Priority +
CLASSWEIGHT * Job.Class.Priority)

All user, group, account, QoS, and class weights are specified by setting the `PRIORITY` attribute of using the respective `*CFG` parameter (namely, `USERCFG`, `GROUPCFG`, `ACCOUNTCFG`, `QOSCFG`, and `CLASSCFG`).

For example, to set user and group priorities, you could use the following:

```
CREDWEIGHT       1
USERWEIGHT       1
GROUPWEIGHT      1
USERCFG[john]    PRIORITY=2000
USERCFG[paul]    PRIORITY=-1000
GROUPCFG[staff]  PRIORITY=10000
```

Class (or queue) priority can also be specified via the resource manager where supported (as in PBS queue priorities). However, if Moab class priority values are also specified, the resource manager priority values will be overwritten.

All priorities can be positive or negative.

4.1.2.C Fairshare (FS) Component

Fairshare components allow you to favor jobs based on short-term historical usage. 5.3 Fairshare describes the configuration and use of fairshare in detail.

The fairshare factor is used to adjust a job's priority based on current and historical percentage system utilization of the job's user, group, account, class, or QoS. This enables sites to steer workload toward a particular usage mix across user, group, account, class, and QoS dimensions.

The fairshare priority factor calculation is as follows:

Priority += FSWEIGHT * MIN(FSCAP, (
  FSUSERWEIGHT * DeltaUserFSUsage +
  FSGROUPWEIGHT * DeltaGroupFSUsage +
  FSACCOUNTWEIGHT * DeltaAccountFSUsage +
  FSQOSWEIGHT * DeltaQOSFSUsage +
  FSCLASSWEIGHT * DeltaClassFSUsage +
  FSJPUWEIGHT * ActiveUserJobs +
  FSPPUWEIGHT * ActiveUserProcs +
  FSPSPUWEIGHT * ActiveUserPS +
  WCACCURACYWEIGHT * UserWCAccuracy)

All `*WEIGHT` parameters just listed are specified on a per partition basis in the `moab.cfg` file. The `Delta*Usage` components represent the difference in actual fairshare usage from the corresponding fairshare usage target. Actual fairshare usage is determined based on historical usage over the time frame specified in the fairshare configuration. The target usage can be a target, floor, or ceiling value as specified in the fairshare configuration file. See 5.3 Fairshare for more information on configuring and tuning fairshare. Additional insight may be available in the fairshare usage example. The `ActiveUser*` components represent current usage by the job's user credential.

How Violated Ceilings and Floors Affect Fairshare-Based Priority

Moab determines `FSUsageWeight` in the previous section. In order to account for violated ceilings and floors, Moab multiplies that number by the `FSUsagePriority` as demonstrated in the following formula:

```
FSPriority = FSUsagePriority * FSUsageWeight
```

When a ceiling or floor is violated, `FSUsagePriority = 0`, so `FSPriority = 0`. This means the job will gain no priority because of fairshare. If fairshare is the only component of priority, then violation takes the priority to 0. For more information, see 5.3.2.C Priority-Based Fairshare and 5.3.2.A Fairshare Targets.

4.1.2.D Resource (RES) Component

Weighting jobs by the amount of resources requested enables you to favor particular types of jobs. Such prioritization may allow you to better meet site mission objectives, improve fairness, or even improve overall system utilization.

Resource based prioritization is valuable when you want to favor jobs based on the resources requested. This is good in three main scenarios: (1) when you need to favor large resource jobs because it's part of your site's mission statement, (2) when you want to level the response time distribution across large and small jobs (small jobs are more easily backfilled and therefore generally have better turnaround time), and (3) when you want to improve system utilization. While this might be surprising, system utilization actually increases as large resource jobs are pushed to the front of the queue. This keeps the smaller jobs in the back where they can be selected for backfill and therefore increase overall system utilization. The situation is like the story about filling a cup with golf balls and sand. If you put the sand in first, it gets in the way and you are unable to put in as many golf balls. However, if you put in the golf balls first, the sand can easily be poured in around them completely filling the cup.

The calculation for determining the total resource priority factor is as follows:

```
Priority += RESWEIGHT* MIN(RESCAP, (
  NODEWEIGHT * TotalNodesRequested +
  PROCWEIGHT * TotalProcessorsRequested +
  MEMWEIGHT * TotalMemoryRequested +
  SWAPWEIGHT * TotalSwapRequested +
  DISKWEIGHT * TotalDiskRequested +
  WALLTIMEWEIGHT* TotalWalltimeRequested +
  PEWEIGHT * TotalPERequested)
```

The sum of all weighted resources components is then multiplied by the `RESWEIGHT` parameter and capped by the `RESCAP` parameter. Memory, Swap, and Disk are all measured in megabytes (MB). The final resource component, PE, represents Processor Equivalents. This component can be viewed as a processor-weighted maximum *percentage of total resources* factor.

For example, if a job requested 25% of the processors and 50% of the total memory on a 128-processor system, it would have a PE value of MAX(25,50) * 128, or 64. The concept of PEs is a highly effective metric in shared resource systems.

Ideal values for requested job processor count and walltime can be specified using the parameters PRIORITYTARGETPROCCOUNT and PRIORITYTARGETDURATION.

4.1.2.E Service (SERVICE) Component

The Service component specifies which service metrics are of greatest value to the site. Favoring one service subcomponent over another generally improves that service metric.

The priority calculation for the service priority factor is as follows:

Priority += SERVICEWEIGHT * (
  QUEUETIMEWEIGHT * <QUEUETIME> +
  XFACTORWEIGHT * <XFACTOR> +
  BYPASSWEIGHT * <BYPASSCOUNT> +
  STARTCOUNTWEIGHT * <STARTCOUNT> +
  DEADLINEWEIGHT * <DEADLINE> +
  SPVIOLATIONWEIGHT * <SPBOOLEAN> +
  USERPRIOWEIGHT * <USERPRIO>)

QueueTime (QUEUETIME) Subcomponent

In the priority calculation, a job's queue time is a duration measured in minutes. Using this subcomponent tends to prioritize jobs in a FIFO order. Favoring queue time improves queue time based fairness metrics and is probably the most widely used single job priority metric. In fact, under the initial default configuration, this is the only priority subcomponent enabled within Moab. It is important to note that within Moab, a job's queue time is not necessarily the amount of time since the job was submitted. The parameter JOBPRIOACCRUALPOLICY enables you to select how a job will accrue queue time based on meeting various throttling policies. Regardless of the policy used to determine a job's queue time, this effective queue time is used in the calculation of the QUEUETIME, XFACTOR, TARGETQUEUETIME, and TARGETXFACTOR priority subcomponent values.

The need for a distinct effective queue time is necessitated by the fact that many sites have users who like to work the system, whatever system it happens to be. A common practice at some long existent sites is for some users to submit a large number of jobs and then place them on hold. These jobs remain with a hold in place for an extended period of time and when the user is ready to run a job, the needed executable and data files are linked into place and the hold released on one of these presubmitted jobs. The extended hold time guarantees that this job is now the highest priority job and will be the next to run. The use of the `JOBPRIOACCRUALPOLICY` parameter can prevent this practice and prevent 'queue stuffers' from doing similar things on a shorter time scale. These 'queue stuffer' users submit hundreds of jobs at once to swamp the machine and consume use of the available compute resources. This parameter prevents the user from gaining any advantage from stuffing the queue by not allowing these jobs to accumulate any queue time based priority until they meet certain idle and active Moab fairness policies (such as max job per user and max idle job per user).

As a final note, you can adjust the QUEUETIMEWEIGHT parameter on a per QoS basis using the QOSCFG parameter and the `QTWEIGHT` attribute. For example, the line `QOSCFG [special] QTWEIGHT=5000` causes jobs using the QoS `special` to have their queue time subcomponent weight increased by 5000.

Expansion Factor (XFACTOR) Subcomponent

The expansion factor subcomponent has an effect similar to the queue time factor but favors shorter jobs based on their requested wallclock run time. In its traditional form, the expansion factor (XFactor) metric is calculated as follows:

XFACTOR = 1 + <QUEUETIME> / <EXECUTIONTIME>

However, a couple of aspects of this calculation make its use more difficult. First, the length of time the job will actually run—<EXECUTIONTIME>—is not actually known until the job completes. All that is known is how much time the job requests. Secondly, as described in the Queue Time Subcomponent section, Moab does not necessarily use the raw time since job submission to determine <QUEUETIME> to prevent various scheduler abuses. Consequently, Moab uses the following modified equation:

XFACTOR = 1 + <EFFQUEUETIME> / <WALLCLOCKLIMIT>

In the equation Moab uses, <EFFQUEUETIME> is the effective queue time subject to the JOBPRIOACCRUALPOLICY parameter and <WALLCLOCKLIMIT> is the user—or system—specified job wallclock limit.

Using this equation, it can be seen that short running jobs will have an XFactor that will grow much faster over time than the xfactor associated with long running jobs. The following table demonstrates this favoring of short running jobs:

Job Queue Time	1 hour	2 hours	4 hours	8 hours	16 hours
XFactor for 1 hour job	1 + (1 / 1) = 2.00	1 + (2 / 1) = 3.00	1 + (4 / 1) = 5.00	1 + (8 / 1) = 9.00	1 + (16 / 1) = 17.0

Job Queue Time	1 hour	2 hours	4 hours	8 hours	16 hours
XFactor for 4 hour job	1 + (1 / 4) = 1.25	1 + (2 / 4) = 1.50	1 + (4 / 4) = 2.00	1 + (8 / 4) = 3.00	1 + (16 / 4) = 5.0

Since XFactor is calculated as a ratio of two values, it is possible for this subcomponent to be almost arbitrarily large, potentially swamping the value of other priority subcomponents. This can be addressed either by using the subcomponent cap XFACTORCAP, or by using the XFMINWCLIMIT parameter. If the latter is used, the calculation for the XFactor subcomponent value becomes:

XFACTOR = 1 + <EFFQUEUETIME> / MAX(<XFMINWCLIMIT>,<WALLCLOCKLIMIT>)

Using the `XFMINWCLIMIT` parameter, enables you to prevent very short jobs from causing the XFactor subcomponent to grow inordinately.

Some sites consider XFactor to be a more fair scheduling performance metric than queue time. At these sites, job XFactor is given far more weight than job queue time when calculating job priority and job XFactor distribution consequently tends to be fairly level across a wide range of job durations. (That is, a flat XFactor distribution of 1.0 results in a one-minute job being queued on average one minute, while a 24-hour job is queued an average of 24 hours.)

Like queue time, the effective XFactor subcomponent weight is the sum of two weights, the `XFACTORWEIGHT` parameter and the QoS-specific XFWEIGHT setting. For example, the line `QOSCFG[special] XFWEIGHT=5000` causes jobs using the QoS `special` to increase their expansion factor subcomponent weight by 5000.

Bypass (BYPASS) Subcomponent

The bypass factor is based on the bypass count of a job where the bypass count is increased by one every time the job is bypassed by a lower priority job via backfill. Backfill starvation has never been reported, but if encountered, use the BYPASS subcomponent.

StartCount (STARTCOUNT) Subcomponent

Apply the startcount factor to sites with trouble starting or completing due to policies or failures. The primary causes of an idle job having a startcount greater than zero are resource manager level job start failure, admin based requeue, or requeue based preemption.

Deadline (DEADLINE) Subcomponent

The deadline factor enables sites to take into consideration the proximity of a job to its DEADLINE. As a jobs moves closer to its deadline its priority increases linearly. This is an alternative to the strict deadline discussed in 6.3.2.B Service Access and Constraints.

Soft Policy Violation (SPVIOLATION) Subcomponent

The soft policy violation factor enables sites to favor jobs that do not violate their associated soft resource limit policies.

User Priority (USERPRIO) Subcomponent

The user priority subcomponent enables sites to consider end-user specified job priority in making the overall job priority calculation. Under Moab, end-user specified priorities can only be negative and are bounded in the range 0 to -1024. See 4.1.4.D User Selectable Prioritization and 4.1.6 Manual Job Priority Adjustment for more information.

User priorities can be positive, ranging from -1024 to 1023, if `ENABLEPOSUSERPRIORITY TRUE` is specified in `moab.cfg`.

4.1.2.F Target Service (TARG) Component

The target factor component of priority takes into account job scheduling performance targets. Currently, this is limited to target expansion factor and target queue time. Unlike the expansion factor and queue time factors described earlier that increase gradually over time, the target factor component is designed to grow exponentially as the target metric is approached. This behavior causes the scheduler to do essentially all in its power to make certain the scheduling targets are met.

The priority calculation for the target factor is as follows:

```
Priority += TARGETWEIGHT* (
  TARGETQUEUETIMEWEIGHT  * QueueTimeComponent +
  TARGETXFACTORWEIGHT    * XFactorComponent)
```

The queue time and expansion factor target are specified on a per QoS basis using the `XFTARGET` and `QTTARGET` attributes with the QOSCFG parameter. The QueueTime and XFactor component calculations are designed to produce small values until the target value begins to approach, at which point these components grow very rapidly. If the target is missed, this component remains high and continues to grow, but it does not grow exponentially.

4.1.2.G Usage (USAGE) Component

The Usage component applies to active jobs only. The priority calculation for the usage priority factor is as follows:

```
Priority += USAGEWEIGHT * (
  USAGECONSUMEDWEIGHT      * ProcSecondsConsumed +
  USAGEHUNGERWEIGHT      * ProcNeededToBalanceDynamicJob +
  USAGEREMAININGWEIGHT     * ProcSecRemaining +
```

USAGEEXECUTIONTIMEWEIGHT * SecondsSinceStart +
USAGEPERCENTWEIGHT * WalltimePercent)

4.1.2.H Job Attribute (ATTR) Component

The Attribute component allows the incorporation of job attributes into a job's priority. The most common usage for this capability is to do one of the following:

- adjust priority based on a job's state (favor suspended jobs)
- adjust priority based on a job's requested node features (favor jobs that request attribute `pvfs`)
- adjust priority based on internal job attributes (disfavor `backfill` or `preemptee` jobs)
- adjust priority based on a job's requested licenses, network consumption, or generic resource requirements

To use job attribute based prioritization, the JOBPRIOF parameter must be specified to set corresponding attribute priorities. To favor jobs based on node feature requirements, the parameter NODETOJOBATTRMAP must be set to map node feature requests to job attributes.

The priority calculation for the attribute priority factor is as follows:

Priority += ATTRWEIGHT * (
ATTRATTRWEIGHT * <ATTRPRIORITY> +
ATTRSTATEWEIGHT * <STATEPRIORITY> +
ATTRGRESWEIGHT * <GRESPRIORITY>
JOBIDWEIGHT * <JOBID> +
JOBNAMEWEIGHT * <JOBNAME_INTEGER>)

Example 4-1:

```
ATTRWEIGHT      100
ATTRATTRWEIGHT    1
ATTRSTATEWEIGHT   1
ATTRGRESWEIGHT    5
# favor suspended jobs
# disfavor preemptible jobs
# favor jobs requesting 'matlab'

JOBPRIOF STATE[Running]=100  STATE[Suspended]=1000  ATTR[PREEMPTEE]=-200  ATTR
[gpfs]=30  GRES[matlab]=400
# map node features to job features

NODETOJOBATTRMAP  gpfs,pvfs
...
```

Related Topics

- 4.2 Node Allocation Policies - PRIORITY
- 2.7.1.A Credential Priority Settings/Priority Weight Offsets
- 10.7 Managing Consumable Generic Resources

4.1.3 Fairshare Job Priority Example

Consider the following information associated with calculating the fairshare factor for job X.

Job X
  User A
  Group B
  Account C
  QOS D
  Class E

User A

Fairshare Target:	50.0
Current Fairshare Usage:	45.0

Group B

Fairshare Target:	[NONE]
Current Fairshare Usage:	65.0

Account C

Fairshare Target:	25.0
Current Fairshare Usage:	35.0

QOS D

Fairshare Target:	10.0+
Current Fairshare Usage:	25.0

Class E

Fairshare Target:	[NONE]
Current Fairshare Usage:	20.0

Priority Weights:

FSWEIGHT	100
FSUSERWEIGHT	10
FSGROUPWEIGHT	20
FSACCOUNTWEIGHT	30
FSQOSWEIGHT	40
FSCLASSWEIGHT	0

In this example, the Fairshare component calculation is this:

```
Priority += 100 * (
        10 * 5 +
        20 * 0 +
        30 * (-10) +
        40 * 0 +
          0 * 0)
```

User A is 5% below his target so fairshare increases the total fairshare factor accordingly. Group B has no target so group fairshare usage is ignored. Account C is above its 10% above its fairshare usage target so this component decreases the job's total fairshare factor. QOS D is 15% over its target but the '+' in the target specification indicates that this is a 'floor' target, only influencing priority when fairshare usage drops below the target value. Therefore, the QOS D fairshare usage delta does not influence the fairshare factor.

Fairshare is a great mechanism for influencing job turnaround time via priority to favor a particular distribution of jobs. However, it is important to realize that fairshare can only favor a particular distribution of jobs, it cannot force it. If user X has a fairshare target of 50% of the machine but does not submit enough jobs, no amount of priority favoring will get user X's usage up to 50%.

Related Topics

- 5.3 Fairshare

4.1.4 Common Priority Usage

In this topic:

Site admins vary widely in their preferred manner of prioritizing jobs. Moab's scheduling hierarchy enables sites to meet job control needs without requiring adjustments to dozens of parameters. Some choose to use numerous subcomponents, others a few, and still others are content with the default FIFO behavior. Any subcomponent that is not of interest can be safely ignored.

4.1.4.A Credential Priority Factors

To help clarify the use of priority weights, a brief example may help. Suppose you wanted to maintain the FIFO behavior but also incorporate some credential based prioritization to favor a special user. Particularly, the site wants the user `john` to receive a higher initial priority than all other users. Configuring this behavior requires two steps. First, the user credential subcomponent must be enabled and second, `john` must have his relative priority specified. Take a look at the sample `moab.cfg` file:

```
USERWEIGHT        1
USERCFG[john]     PRIORITY=300
```

The 'USER' priority subcomponent was enabled by setting the USERWEIGHT parameter. In fact, the parameters used to specify the weights of all components and subcomponents follow this same '*WEIGHT' naming convention (as in RESWEIGHT and TARGETQUEUETIMEWEIGHT).

The second part of the example involves specifying the actual user priority for the user `john`. This is accomplished using the USERCFG parameter. Why was the priority 300 selected and not some other value? Is this value arbitrary? As in any priority system, actual priority values are meaningless, only relative values are important. In this case, we are required to balance user priorities with the default queue time based priorities. Since queuetime priority is measured in minutes queued, the user priority of 300 places a job by user `john` on par with a job submitted 5 minutes earlier by another user.

Is this what the site wants? Maybe, maybe not. At the onset, most sites are uncertain what they want in prioritization. Often, an estimate initiates prioritization and adjustments occur over time. Cluster resources evolve, the workload evolves, and even site policies evolve, resulting in changing priority needs over time. Anecdotal evidence indicates that most sites establish a relatively stable priority policy within a few iterations and make only occasional adjustments to priority weights from that point.

4.1.4.B Service Level Priority Factors

In another example, suppose a site admin wants to do the following:

- Favor jobs in the low, medium, and high QoSes so they will run in QoS order
- Balance job expansion factor
- Use job queue time to prevent jobs from starving

Under such conditions, the sample `moab.cfg` file might appear as follows:

```
QOSWEIGHT              1
XFACTORWEIGHT          1
```

```
QUEUETIMEWEIGHT        10
TARGETQUEUETIMEWEIGHT   1
QOSCFG[low]       PRIORITY=1000
QOSCFG[medium]    PRIORITY=10000
QOSCFG[high]      PRIORITY=100000
QOSCFG[DEFAULT]   QTTARGET=4:00:00
```

This example is a bit more complicated but is more typical of the needs of many sites. The desired QoS weightings are established by enabling the QoS subfactor using the QOSWEIGHT parameter while the various QoS priorities are specified using QOSCFG. XFACTORWEIGHT is then set as this subcomponent tends to establish a balanced distribution of expansion factors across all jobs. Next, the queuetime component is used to gradually raise the priority of all jobs based on the length of time they have been queued. Note that in this case, the parameter QUEUETIMEWEIGHT was explicitly set to 10, overriding its default value of 1. Finally, the TARGETQUEUETIMEWEIGHT parameter is used in conjunction with the USERCFG line to specify a queue time target of 4 hours.

4.1.4.C Priority Factor Caps

Assume now that the site admin is content with this priority mix but has a problem with users submitting large numbers of very short jobs. Very short jobs tend to have rapidly growing XFactor values and will consequently quickly jump to the head of the queue. In this case, a factor cap would be appropriate. Such caps allow you to limit the contribution of a job's priority factor to be within a defined range. This prevents certain priority factors from swamping others. Caps can be applied to either priority components or subcomponents and are specified using the `<COMPONENTNAME>CAP` parameter (such as `QUEUETIMECAP`, `RESCAP`, and `SERVCAP`). Note that both component and subcomponent caps apply to the preweighted value, as in the following equation:

```
Priority =
      C1WEIGHT * MIN(C1CAP,SUM(
        S11WEIGHT * MIN(S11CAP,S11S) +
        S12WEIGHT * MIN(S12CAP,S12S) +
        ...)) +
      C2WEIGHT * MIN(C2CAP,SUM(
        S21WEIGHT * MIN(S21CAP,S21S) +
        S22WEIGHT * MIN(S22CAP,S22S) +
        ...)) +
      ...
```

Example 4-2: Priority cap

```
QOSWEIGHT              1
QOSCAP                 10000
XFACTORWEIGHT          1
XFACTORCAP             1000
QUEUETIMEWEIGHT        10
QUEUETIMECAP           1000
```

4.1.4.D User Selectable Prioritization

Moab allows users to specify a job priority to jobs they own or manage. This priority can be set at job submission time or it can be dynamically modified (using setspri or mjobctl) after submitting the job. For fairness reasons, users can only apply a negative priority to their job and therefore slide it further back in the queue. This enables users to allow their more important jobs to run before their less important ones without gaining unfair advantage over other users.

User priorities can be positive if `ENABLEPOSUSERPRIORITY TRUE` is specified in `moab.cfg`.

In order to set `ENABLEPOSUSERPRIORITY`, you must change the USERPRIOWEIGHT parameter from its default value of 0. For example:

```
USERPRIOWEIGHT   100
```

```
> setspri -r 100 332411
successfully modified job priority
```

Specifying a user priority at job submission time is resource manager specific. See the associated resource manager documentation for more information.

User Selectable Priority w/QoS

Using the QoS facility, organizations can set up an environment where users can more freely select the desired priority of a given job. Organizations can enable access to a number of QoSes each with its own charging rate, priority, and target service levels. Users can then assign job importance by selecting the appropriate QoS. If desired, this can allow a user to jump ahead of other users in the queue if they are willing to pay the associated costs.

Related Topics

- 4.1.6 Manual Job Priority Adjustment

4.1.5 Prioritization Strategies

Each component or subcomponent can be used to accomplish different objectives. `WALLTIME` can be used to favor (or disfavor) jobs based on their duration. Likewise, `ACCOUNT` can be used to favor jobs associated with a particular project while `QUEUETIME` can be used to favor those jobs waiting the longest.

- Queue Time
- Expansion Factor
- Resource
- Fairshare
- Credential
- Target Metrics

Each priority factor group can contain one or more subfactors. For example, the Resource factor consists of Node, Processor, Memory, Swap, Disk, and PE components. From the table in the Job Priority Factors section, it is apparent that the prioritization problem is fairly complex since every site needs to prioritize a bit differently. When calculating a priority, the various priority factors are summed and then bounded between 0 and MAX_PRIO_VAL, which is currently defined as 100000000 (one billion).

The mdiag -p command assists with visualizing the priority distribution resulting from the current job priority configuration. Also, the showstats -f command helps indicate the impact of the current priority settings on scheduler service distributions.

4.1.6 Manual Job Priority Adjustment

Batch administrators regularly find a need to adjust the calculated priority of a job to meet current needs. Current needs often are broken into two categories:

1. The need to run an administrator test job as soon as possible.
2. The need to pacify a disserviced user.

You can use the setspri command to handle these issues in one of two ways; this command allows the specification of either a relative priority adjustment or the specification of an absolute priority. Using absolute priority specification, admins can set a job priority guaranteed to be higher than any calculated value. Where Moab-calculated job priorities are in the range of 0 to 1 billion, system admin assigned absolute priorities start at 1 billion and go up. Issuing the `setspri <PRIO> <JOBID>` command, for example, assigns a priority of 1 billion + <PRIO> to the job. Therefore, `setspri 5 job.1294` sets the priority of 'job.1294' to 1000000005.

For more information, see 4.1.4.D User Selectable Prioritization.

4.2 Node Allocation Policies

While job prioritization enables you to determine which job to run, node allocation policies allow you to specify how available resources should be allocated to each job. The algorithm used is specified by the parameter NODEALLOCATIONPOLICY. There are multiple node allocation policies to choose from allowing selection based on reservation constraints, node configuration, resource usage, preferences, and other factors. You can specify these policies with a system-wide default value, on a per-partition basis, or on a per-job basis. Note that `LASTAVAILABLE` is the default policy.

Available algorithms are described in detail in the following sections and include: CONTIGUOUS, CPULOAD, FIRSTAVAILABLE, LASTAVAILABLE, MINRESOURCE, MAXBALANCE, PLUGIN, PRIORITY.

In this section:

4.2.1 Node Allocation Overview

Node allocation is the process of selecting the best resources to allocate to a job from a list of available resources. Making this decision intelligently is important in an environment that possesses one or more of the following attributes:

- Heterogeneous resources (resources which vary from node to node in terms of quantity or quality)
- Shared nodes (nodes can be utilized by more than one job)
- Reservations or service guarantees
- Non-flat network (a network where a perceptible performance degradation may potentially exist depending on workload placement)

In this topic:

4.2.1.A Heterogeneous Resources

Moab analyzes job processing requirements and assigns resources to maximize hardware utility.

For example, suppose two nodes are available in a system, A and B. Node A has 768 MB of RAM and node B has 512 MB. The next two jobs in the queue are X and Y. Job X requests 256 MB and job Y requests 640 MB. Job X is next in the queue and can fit on either node, but Moab recognizes that job Y (640 MB) can only fit on node A (768 MB). Instead of putting job X on node A and blocking job Y, Moab can put job X on node B and job Y on node A.

4.2.1.B Shared Nodes

Symmetric Multiprocessing (SMP)

When sharing SMP-based compute resources amongst tasks from more than one job, resource contention and fragmentation issues arise. In SMP environments, the general goal is to deliver maximum system utilization for a combination of compute-intensive and memory-intensive jobs while preventing overcommitment of resources.

By default, most current systems do not do a good job of logically partitioning the resources (such as CPU, memory, and network bandwidth) available on a given node. Consequently contention often arises between tasks of independent jobs on the node. This can result in a slowdown for all jobs involved, which can have significant ramifications if large-way parallel jobs are involved. Virtualization, CPU sets, and other techniques are maturing quickly as methods to provide logical partitioning within shared resources.

On large-way SMP systems (> 32 processors/node), job packing can result in intra-node fragmentation. For example, take two nodes, A and B, each with 64 processors. Assume they are currently loaded with various jobs and A has 24 and B has 12 processors free. Two jobs are submitted; job X requests 10 processors and job Y requests 20 processors. Job X can start on either node but starting it on node A prevents job Y from running. An algorithm to handle intra-node fragmentation is straightforward for a single resource case, but the algorithm becomes more involved when jobs request a combination of processors, memory, and local disk. These workload factors should be considered when selecting a site's node allocation policy, and identifying appropriate policies for handling resource utilization limit violations.

Interactive Nodes

In many cases, sites are interested in allowing multiple users to simultaneously use one or more nodes for interactive purposes. Workload is commonly not compute intensive consisting of intermittent tasks including coding, compiling, and testing. Because these jobs are highly variant in terms of resource usage over time, sites are able to pack a larger number of these jobs onto the same node. Consequently, a common practice is to restrict job scheduling based on utilized, rather than dedicated resources.

Interactive Node Example

The example configuration files that follow show one method by which node sharing can be accomplished within a Torque + Moab environment. This example is based on a hypothetical cluster composed of 4 nodes each with 4 cores. For the compute nodes, job tasks are limited to actual cores preventing overcommitment of resources. For the interactive nodes, up to 32 job tasks are allowed, but the node also stops allowing additional tasks if either memory is fully utilized or if the CPU load exceeds 4.0. Therefore, Moab continues packing the interactive nodes with jobs until carrying capacity is reached.

Example 4-3: /opt/moab/etc/moab.cfg

```
# constrain interactive jobs to interactive nodes
# constrain interactive jobs to 900 proc-seconds
CLASSCFG[interactive]  HOSTLIST=interactive01,interactive02
CLASSCFG[interactive]  MAX.CPUTIME=900
RESOURCELIMITPOLICY    CPUTIME:ALWAYS:CANCEL
# base interactive node allocation on load and jobs
NODEALLOCATIONPOLICY  PRIORITY
NODECFG[interactive01] PRIORITYF='-20*LOAD - JOBCOUNT'
NODECFG[interactive02] PRIORITYF='-20*LOAD - JOBCOUNT'
```

Example 4-4: /var/spool/torque/server_priv/nodes

```
interactive01 np=32
interactive02 np=32
compute01     np=4
compute02     np=4
```

Example 4-5: /var/spool/torque/mom_priv/config on interactive01

```
# interactive01
$max_load 4.0
```

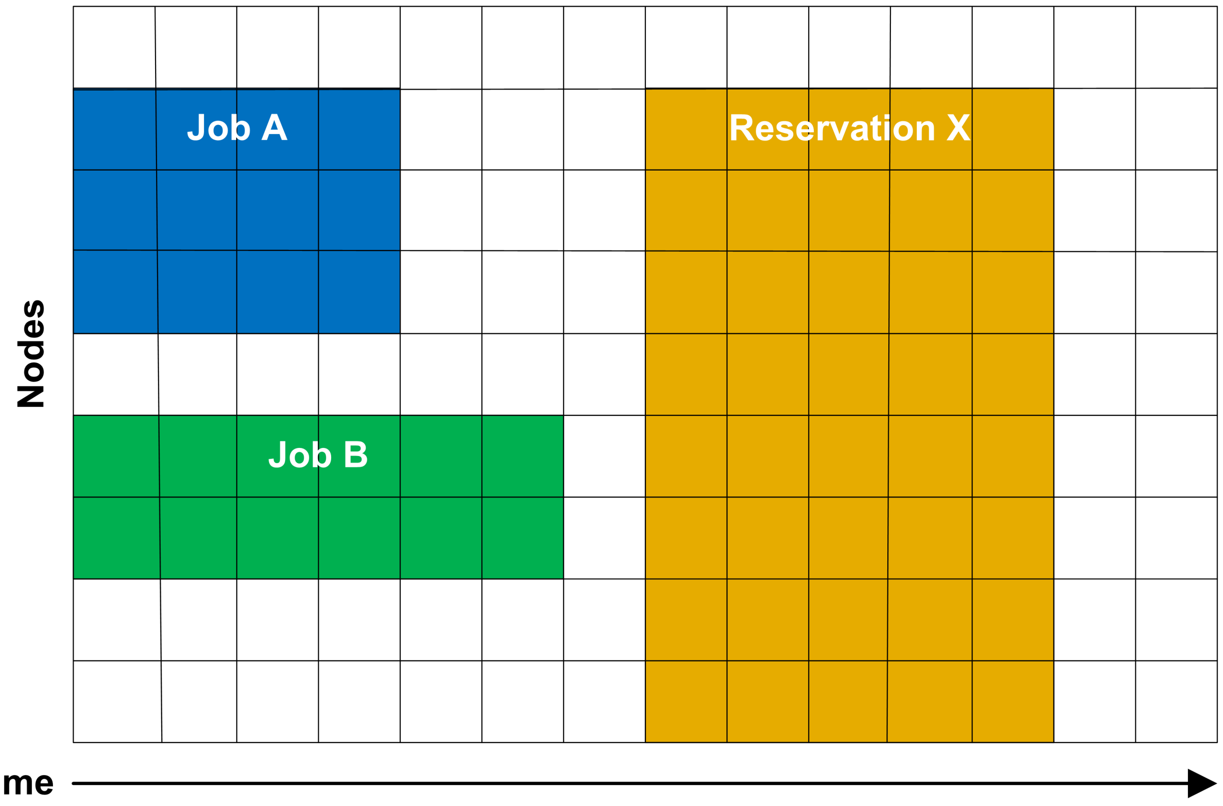
Example 4-6: /var/spool/torque/mom_priv/config on interactive02

```
# interactive02
$max_load 4.0
```

4.2.1.C Reservations or Service Guarantees

A reservation-based system adds the time dimension into the node allocation decision. With reservations, node resources must be viewed in a type of two dimension node-time space. Allocating nodes to jobs fragments this node-time space and makes it more difficult to schedule jobs in the remaining, more constrained node-time slots. Allocation decisions should be made in such a way as to minimize this fragmentation and maximize the scheduler's ability to continue to start jobs in existing slots. The following figure shows that job A and job B are running. A reservation, X, is created some time in the future. Assume that job A is 2 hours long and job B is 3 hours long. Again, two new single-processor jobs are submitted, C and D; job C requires 3 hours of compute time while job D requires 5 hours. Either job will just fit in the free space located above job A or in the free space located below job B. If job C is placed above job A, job D, requiring 5 hours of time will be prevented from running by the presence of reservation X. However, if job C is placed below job B, job D can still start immediately above job A.

Image 4-1: Job A, Job B, and Reservation X scheduled on nodes



The preceding example demonstrates the importance of time based reservation information in making node allocation decisions, both at the time of starting jobs and at the time of creating reservations. The impact of time based issues grows significantly with the number of reservations in place on a given system. The `LASTAVAILABLE` algorithm works on this premise, locating resources that have the smallest space between the end of a job under consideration and the start of a future reservation.

4.2.1.D Non-Flat Network

On systems where network connections do not resemble a flat all-to-all topology, task placement may impact performance of communication intensive parallel jobs. If latencies and network bandwidth between any two nodes vary significantly, the node allocation algorithm should attempt to pack tasks of a given job as close to each other as possible to minimize impact of bandwidth and latency differences.

4.2.2 Node Selection Factors

While the node allocation policy determines which nodes a job will use, other factors narrow the options before the policy makes the final decision. The following process demonstrates how Moab executes its node allocation process and how other policies affect the decision:

1. Moab eliminates nodes that do not meet the hard resource requirements set by the job.
2. Moab gathers affinity information, first from workload proximity rules and then from reservation affinity rules (see the section Affinity for more information). Reservation affinity rules trump workload proximity rules.
3. Moab allocates nodes using the allocation policy:
    - If more than enough nodes with Required affinity exist, only they are passed down for the final sort by the node allocation policy.
    - If the number of nodes with Required affinity matches the number of nodes requested exactly, then the node allocation policy is skipped entirely and all of those nodes are assigned to the job.
    - If too few nodes have Required affinity, all of them are assigned to the job, then the node allocation policy is applied to the remaining eligible nodes (after Required, Moab will use Positive, then Neutral, then Negative).

4.2.3 Resource-Based Algorithms

Moab contains a number of allocation algorithms that address some of the needs described earlier. You can also create allocation algorithms and interface them with the Moab scheduling system. Each of these policies has a name and descriptive alias. They can be configured using either one, but Moab will only report their names.

If the parameter ENABLEHIGHTHROUGHPUT is `TRUE`, you must set the parameter NODEALLOCATIONPOLICY to `FIRSTAVAILABLE`.

The current suite of algorithms is described in the table below:

Allocation Algorithms

Allocation Algorithm Name	Alias	Description
CONTIGUOUS	Contiguous	Allocates nodes in contiguous (linear) blocks as required by the Compaq RMS system.
CPULOAD	ProcessorLoad	Nodes are selected that have the maximum amount of available, unused CPU power (<#of CPUs> - <CPU load>). CPULOAD is a good algorithm for timesharing node systems and applies to jobs starting immediately. For the purpose of future reservations, the MINRESOURCE algorithm is used.
FIRSTAVAILABLE	InReportedOrder	Simple first come, first served algorithm where nodes are allocated in the order they are presented by the resource manager. This is a very simple, and very fast algorithm.
LASTAVAILABLE	InReserveReportedOrder	Nodes are allocated in descending order that they are presented by the resource manager, or the reverse of FIRSTAVAILABLE.
MAXBALANCE	ProcessorSpeedBalance	Attempts to allocate the most balanced set of nodes possible to a job. In most cases, but not all, the metric for balance of the nodes is node procspeed. Therefore, if possible, nodes with identical procspeeds are allocated to the job. If identical procspeed nodes cannot be found, the algorithm allocates the set of nodes with the minimum node procspeed span or range.
MINRESOURCE	MinimumConfiguredResources	Prioritizes nodes according to the configured memory resources on each node. Those nodes with the fewest configured memory resources, that still meet the job's resource constraints, are selected.
PRIORITY	CustomPriority	Enables you to specify the priority of various static and dynamic aspects of compute nodes and allocate them with preference for higher priority nodes. It is highly flexible allowing node attribute and usage information to be combined with reservation affinity. Using node allocation

Allocation Algorithm Name	Alias	Description
		priority, you can specify the following priority components: • ADISK - Local disk currently available to batch jobs in MB. • AMEM - Real memory currently available to batch jobs in MB. • APROCS - Processors currently available to batch jobs on node (configured procs - dedicated procs). • ARCH[<ARCH>] - Processor architecture.
		• ASWAP - Virtual memory currently available to batch jobs in MB. • CDISK - Total local disk allocated for use by batch jobs in MB. • CMEM - Total real memory on node in MB. • COST - Based on node CHARGERATE. • CPROCS - Total processors on node.
		• CSWAP - Total virtual memory configured on node in MB. • FEATURE[<FNAME>] - Boolean; specified feature is present on node. • FREETIME - FREETIME is calculated as the time during which there is no reservation on the machine. It uses either the job wallclock limit (if there is a job), or 2 months. The more free time a node has within either the job wallclock limit or 2 months, the higher this value will be. • GMETRIC[<GMNAME>] - Current value of specified generic metric on node. • JOBCOUNT - Number of jobs currently running on node.
		• JOBFREETIME - The number of seconds that the node is idle between now and when the job is scheduled to start. • LOAD - Current 1 minute load average. • MTBF - Mean time between failures (in seconds).

Allocation Algorithm Name	Alias	Description
		• NODEINDEX - Node's nodeindex as specified by the resource manager. • OS - True if job compute requirements match node operating system.
		• PARAPROCS - Processors currently available to batch jobs within partition (configured procs - dedicated procs). • POWER - TRUE if node is ON. • PREF - Boolean; node meets job specific resource preferences. • PRIORITY - Administrator specified node priority. • RANDOM - Per iteration random value between 0 and 1. (Allows introduction of random allocation factor.) Regardless of coefficient, the contribution of this weighted factor cannot exceed 32768. The coefficient, if any, of the RANDOM component must precede, not follow, the component in order to work correctly. For example: `100 * RANDOM`
		• SPEED - If set, node processor speed (procspeed); otherwise, relative node speed. • SUSPENDEDJCOUNT - Number of suspended jobs currently on the node. • USAGE - Percentage of time node has been running batch jobs since the last statistics initialization. • WINDOWTIME - The window of time between the end of one reservation and the beginning of another. This algorithm, given a negative value, can be used to pack reservations as close together on a node as possible.
		The node allocation priority function can be

Allocation Algorithm Name	Alias	Description
		specified on a node by node or cluster wide basis. In both cases, the recommended approach is to specify the PRIORITYF attribute with the NODECFG parameter. Some examples follow.

		Example 1: Favor the fastest nodes with the most available memory that are running the fewest jobs.
		`NODEALLOCATIONPOLICY PRIORITY`
		`NODECFG[DEFAULT] PRIORITYF='SPEED + .01 * AMEM - 10 * JOBCOUNT'`
		`...`
		If spaces are placed within the priority function for readability, the priority function value must be quoted to allow proper parsing.

		Example 2: Favor the nodes with the least amount of idle time between now and the job's scheduled start time.
		`NODEALLOCATIONPOLICY PRIORITY`
		`NODECFG[DEFAULT] PRIORITYF=-JOBFREETIME`
		Moab stacks jobs on the nodes that are busiest between now and the job's scheduled start time.

		Example 3: A site has a batch system consisting of two dedicated 'batchX' nodes, and numerous desktop systems. The allocation function should favor batch nodes first, followed by desktop systems that are the least loaded and have received the least historical usage.
		`NODEALLOCATIONPOLICY PRIORITY`
		`NODECFG[DEFAULT] PRIORITYF='-LOAD - 5*USAGE'`
		`NODECFG[batch1] PRIORITY=1000 PRIORITYF='PRIORITY + APROCS'`
		`NODECFG[batch2] PRIORITY=1000 PRIORITYF='PRIORITY + APROCS'`
		`...`

Allocation Algorithm Name	Alias	Description

		Example 4: Pack tasks onto loaded nodes first.
		`NODEALLOCATIONPOLICY PRIORITY`
		`NODECFG[DEFAULT] PRIORITYF=JOBCOUNT`
		`...`

		Example 5: Pack tasks onto nodes with the most processors available and the lowest CPU temperature.
		`RMCFG[torque] TYPE=pbs`
		`RMCFG[temp]   TYPE=NATIVE`
		`CLUSTERQUERYURL=exec://$TOOLSDIR/hwmon.pl`
		`NODEALLOCATIONPOLICY PRIORITY`
		`NODECFG[DEFAULT] PRIORITYF='100*APROCS - GMETRIC [temp]'`
		`...`

4.2.4 User-Defined Algorithms

User-defined algorithms allow admins to define their own algorithms based on factors such as their system's network topology. When node allocation is based on topology, jobs finish faster, admins see better cluster productivity and users pay less for resources.

PLUGIN

This algorithm enables admins to define their own node allocation policy and create a plug-in that allocates nodes based on factors such as a cluster's network topology. This has the following advantages:

- plug-ins keep the source code of the cluster's interconnect network for node allocation separate from Moab's source code (customers can implement plug-ins independent of Moab's release schedule)
- plug-ins can be independently created and tailored to specific hardware and network topology
- plug-ins can be modified without assistance from Adaptive Computing

4.2.5 Specifying *Per Job* Resource Preferences

While the resource based node allocation algorithms can make a good guess at what compute resources would best satisfy a job, sites often possess a subset of jobs that benefit from more explicit resource allocation specification. For example, one job might perform best on a particular subset of nodes due to direct access to a tape drive, another might be very memory intensive. Resource preferences are distinct from node requirements. While the former describes what a job needs to run at all, the latter describes what the job needs to run well. In general, a scheduler must satisfy a job's node requirement specification and then satisfy the job's resource preferences as best as possible.

In this topic:

4.2.5.A Specifying Resource Preferences

A number of resource managers natively support the concept of resource preferences (such as Loadleveler). When using these systems, the language specific preferences keywords can be used. For systems that do not support resource preferences natively, Moab provides a resource manager extension keyword 'PREF', which you can use to specify desired resources. This extension allows specification of node features, memory, swap, and disk space conditions that define whether the node is considered preferred.

4.2.5.B Selecting Preferred Resources

Enforcing resource preferences is not completely straightforward. A site might have a number of potentially conflicting requirements that the scheduler is asked to simultaneously satisfy. For example, a scheduler may be asked to maximize the proximity of the allocated nodes at the same time it is supposed to satisfy resource preferences and minimize node overcommitment. To allow site specific weighting of these varying requirements, Moab allows resource preferences to be enabled through the PRIORITY node allocation algorithm. For example, to use resource preferences together with node load, the following configuration might be used:

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT]     PRIORITYF='5 * PREF - LOAD'
...
```

To request specific resource preferences, a user could then submit a job indicating those preferences. In the case of a PBS job, the following can be used:

```
> qsub -l nodes=4,walltime=1:00:00,pref=feature:fast
```

Related Topics

- 10.8 Enabling Generic Metrics
- NALLOCPOLICY - per job node allocation policy specification

4.3 Node Access Policies

Moab allocates resources to jobs on the basis of a job task—an atomic collection of resources that must be co-located on a single compute node. A given job may request 20 tasks where each task is defined as 1 processor and 128 MB of RAM. Compute nodes with multiple processors often possess enough resources to support more than one task simultaneously. When it is possible for more than one task to run on a node, node access policies determine which tasks can share the compute node's resources.

In this section:

4.3.1 Node Access Policy Descriptions

Moab supports a distinct number of node access policies, which are listed in the following table:

Policy	Description
SHARED	Tasks from any combination of jobs can use available resources.
SHAREDONLY	Only jobs requesting shared node access can use available resources.
SINGLEACCOUNT	Tasks from any jobs owned by the same account can use available resources.
SINGLECLASS	Tasks from any jobs owned by the same class can use available resources.
SINGLEGROUP	Tasks from any jobs owned by the same group can use available resources.
SINGLEJOB	Only tasks from a single job can use the node's resources.

Policy	Description
	When enforcing limits using CLASSCFG attributes, use MAX.NODE instead of MAX.PROC. MAX.PROC enforces the requested processors, not the actual processors dedicated to the job.
SINGLETASK	Only a single task from a single job can run on the node.
SINGLEUSER	Tasks from any jobs owned by the same user can use available resources.
UNIQUEUSER	Any number of tasks from a single job can allocate resources from a node but only if the user has no other jobs running on that node. UNIQUEUSER limits the number of jobs a single user can run on a node, allowing other users to run jobs with the remaining resources. This policy is useful in environments where job epilog/prologs scripts are used to clean up processes based on userid.

4.3.2 Configuring Node Access Policies

The global node access polices can be specified via the parameter NODEACCESSPOLICY. This global default can be overridden on a per node basis with the ACCESS attribute of the NODECFG parameter or on a per job basis using the resource manager extension NACCESSPOLICY. Finally, a per queue node access policy can also be specified by setting either the NODEACCESSPOLICY or FORCENODEACCESSPOLICY attributes of the CLASSCFG parameter. FORCENODEACCESSPOLICY overrides any per job specification in all cases, whereas NODEACCESSPOLICY is overridden by per job specification.

When multiple node access policies apply to a given job or node (for example SINGLEJOB is configured globally but the class is configured as SHARED) then the more restrictive policy applies. The most restrictive policy is SINGLETASK, followed by SINGLEJOB, the single credentials, and SHARED being the least restrictive.

By default, nodes are accessible using the setting of the system wide NODEACCESSPOLICY parameter unless a specific ACCESS policy is specified on a per node basis using the NODECFG parameter. Jobs can override this policy and subsequent jobs are bound to conform to the access policies of all jobs currently running on a given node. For example, if the NODEACCESSPOLICY parameter is set to SHARED, a new job can be launched on an idle node with a job specific access policy of SINGLEUSER. While this job runs, the effective node access policy changes to SINGLEUSER and subsequent job tasks can only be launched on this node provided they are submitted by the same user.

When all single user jobs have completed on that node, the effective node access policy reverts back to SHARED and the node can again be used in SHARED mode.

For example, to set a global policy of SINGLETASK on all nodes except nodes 13 and 14, use the following:

```
# by default, enforce dedicated node access on all nodes
NODEACCESSPOLICY  SINGLETASK
# allow nodes 13 and 14 to be shared
NODECFG[node13]   ACCESS=SHARED
NODECFG[node14]   ACCESS=SHARED
```

You can also set SINGLEJOB using the qsub node-exclusive option (-n). For example:

```
qsub -n jobscript.sh
```

This will set node_exclusive = True in the output for qstat -f <job Id>.

Alternatively, you could also use either of the following:

```
qsub -l naccesspolicy=singlejob jobscript.sh
qsub -W x=naccesspolicy:singlejob jobscript.sh
```

Related Topics

- 11.3 Resource Manager Extensions - NALLOCPOLICY (per job naccesspolicy specification)
- JOBNODEMATCHPOLICY parameter
- NODEAVAILABILITYPOLICY parameter

4.4 Node Availability Policies

In this section:

Moab enables several features relating to node availability. These include policies that determine how per node resource availability should be reported, how node failures are detected, and what should be done in the event of a node failure.

4.4.1 Node Resource Availability Policies

Moab allows a job to be launched on a given compute node as long as the node is not full or busy. The NODEAVAILABILITYPOLICY parameter enables you to determine what criteria constitute a node being busy. The valid settings are listed in the following table:

Availability Policy	Description
DEDICATED	The node is considered busy if dedicated resources equal or exceed configured resources.
UTILIZED	The node is considered busy if utilized resources equal or exceed configured resources.
COMBINED	The node is considered busy if either dedicated or utilized resources equal or exceed configured resources.

The default setting for all nodes is `COMBINED`, indicating that a node can accept workload so long as the jobs that the node was allocated to do not request or use more resources than the node has available. In a load balancing environment, this might not be the desired behavior. Setting the `NODEAVAILABILITYPOLICY` parameter to `UTILIZED` allows jobs to be packed onto a node even if the aggregate resources requested exceed the resources configured. For example, assume a scenario with a 4-processor compute node and 8 jobs requesting 1 processor each. If the resource availability policy was set to `COMBINED`, this node would only allow 4 jobs to start on this node even if the jobs induced a load of less than 1.0 each. With the resource availability policy set to `UTILIZED`, the scheduler continues allowing jobs to start on the node until the node's load average exceeds a per processor load value of 1.0 (in this case, a total load of 4.0). To prevent a node from being over populated within a single scheduling iteration, Moab artificially raises the node's load for one scheduling iteration when starting a new job. On subsequent iterations, the actual measured node load information is used.

Per Resource Availability Policies

By default, the `NODEAVAILABILITYPOLICY` sets a global per node resource availability policy. This policy applies to all resource types on each node such as processors, memory, swap, and local disk. However, the syntax of this parameter is as follows: `<POLICY> [:<RESOURCETYPE>] ...`

This syntax allows per resource availability specification. For example, consider the following:

```
NODEAVAILABILITYPOLICY  DEDICATED:PROC COMBINED:MEM COMBINED:DISK
...
```

This configuration causes Moab to only consider the quantity of processing resources actually dedicated to active jobs running on each node and ignore utilized processor information (such as CPU load). For memory and disk, both utilized resource information and dedicated resource information should be combined to determine what resources are actually available for new jobs.

4.4.2 Node Categorization

Moab allows organizations to detect and use far richer information regarding node status than the standard batch 'idle,' 'busy,' 'down states' commonly found. Using node categorization, organizations can record, track, and report on per node and cluster level status including the following categories:

Category	Description
Active	Node is healthy and currently executing batch workload.
BatchFailure	Node is unavailable due to a failure in the underlying batch system (such as a resource manager server or resource manager node daemon).
Benchmark	Node is reserved for benchmarking.
EmergencyMaintenance	Node is reserved for unscheduled system maintenance.
GridReservation	Node is reserved for grid use.
HardwareFailure	Node is unavailable due to a failure in one or more aspects of its hardware configuration (such as a power failure, excessive temperature, memory, processor, or swap failure).
HardwareMaintenance	Node is reserved for scheduled system maintenance.
Idle	Node is healthy and is currently not executing batch workload.
JobReservation	Node is reserved for job use.
NetworkFailure	Node is unavailable due to a failure in its network adapter or in the switch.

Category	Description
Other	Node is in an uncategorized state.
OtherFailure	Node is unavailable due to a general failure.
PersonalReservation	Node is reserved for dedicated use by a personal reservation.
Site[1-8]	Site specified usage categorization.
SoftwareFailure	Node is unavailable due to a failure in a local software service (such as automounter, security or information service such as NIS, local databases, or other required software services).
SoftwareMaintenance	Node is reserved for software maintenance.
StandingReservation	Node is reserved by a standing reservation.
StorageFailure	Node is unavailable due to a failure in the cluster storage system or local storage infrastructure (such as failures in Lustre, GPFS, PVFS, or SAN).
UserReservation	Node is reserved for dedicated use by a particular user or group and may or may not be actively executing jobs.

Node categories can be explicitly assigned by cluster administrators using the mrsvctl -c command to create a reservation and associate a category with that node for a specified timeframe. Further, outside of this explicit specification, Moab automatically mines all configured interfaces to learn about its environment and the health of the resources it is managing. Consequently, Moab can identify many hardware failures, software failures, and batch failures without any additional configuration. However, it is often desirable to make additional information available to Moab to allow it to integrate this information into reports; automatically notify managers, users, and admins; adjust internal policies to steer workload around failures; and launch various custom triggers to rectify or mitigate the problem.

You can specify the FORCERSVSUBTYPE parameter to require all administrative reservations be associated with a node category at reservation creation time. For example:

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
FORCERSVSUBTYPE  TRUE
```

Node health and performance information from external systems can be imported into Moab using the Native Resource Manager interface. This is commonly done using generic metrics or consumable generic resources for performance and node categories or node variables for status information. Combined with arbitrary node messaging information, Moab can combine detailed information from remote services and report this to other external services.

Use the NODECATCREDLIST parameter to generate extended node category based statistics.

4.4.3 Node Failure/Performance Based Notification

Moab can be configured to cause node failures and node performance levels that cross specified thresholds to trigger notification events. This is accomplished using the GEVENTCFG parameter as described in the section Generic Event Overview. For example, the following configuration can be used to trigger an email to admins each time a node is marked down:

```
GEVENTCFG[nodedown] ACTION=notify REARM=00:20:00
...
```

4.4.4 Node Failure/Performance Based Triggers

Moab supports per node triggers that can be configured to fire when specific events are fired or specific thresholds are met. These triggers can be used to modify internal policies or take external actions. A few examples follow:

- decrease node allocation priority if node throughput drops below threshold X
- launch local diagnostic/recovery script if parallel file system mounts become stale
- reset high performance network adapters if high speed network connectivity fails
- create general system reservation on node if processor or memory failure occurs

As mentioned, Moab triggers can be used to initiate almost any action, from sending mail to updating a database, to publishing data for an SNMP trap, to driving a web service.

4.4.5 Handling Transient Node Failures

Since Moab actively schedules both current and future actions of the cluster, it is often important for it to have a reasonable estimate of when failed nodes will be again available for use. This knowledge is particularly useful for proper scheduling of new jobs and management of resources in regard to backfill. With backfill, Moab determines which resources are available for priority jobs and when the highest priority idle jobs can run. If a node experiences a failure, Moab should have a concept of when this node will be restored.

When Moab analyzes down nodes for allocation, one of two issues may occur with the highest priority jobs. If Moab believes that down nodes will not be recovered for an extended period of time, a transient node failure within a reservation for a priority job might cause the reservation to slide far into the future allowing other lower priority jobs to allocate and launch on nodes previously reserved for it. Moments later, when the transient node failures are resolved, Moab might be unable to restore the early reservation start time as other jobs may already have been launched on previously available nodes.

In the reverse scenario, if Moab recognizes a likelihood that down nodes will be restored too quickly, it might make reservations for top priority jobs that allocate those nodes. Over time, Moab slides those reservations further into the future as it determines that the reserved nodes are not being recovered. While this does not delay the start of the top priority jobs, these unfulfilled reservations can end up blocking other jobs that should have properly been backfilled and executed.

In this topic:

4.4.5.A Creating Automatic Reservations

If a node experiences occasional transient failures (often not associated with a node state of down), Moab can automatically create a temporary reservation over the node to allow the transient failure time to clear and prevent Moab from attempting to re-use the node while the failure is active. This reservation behavior is controlled using the NODEFAILURERESERVETIME parameter as in the following example:

```
# reserve nodes for 1 minute if transient failures are detected
NODEFAILURERESERVETIME  00:01:00
```

4.4.5.B Blocking Out Down Nodes

If one or more resource managers identify failures and mark nodes as down, Moab can be configured to associate a default *unavailability* time with this failure and the node state *down*. This is accomplished using the NODEDOWNSTATEDELAYTIME parameter. This delay time floats and is measured as a fixed time into the future from the time 'NOW'; it is not associated with the time the node was originally marked down. For example, if the delay time was set to 10 minutes, and a node was marked down 20 minutes ago, Moab would still consider the node unavailable until 10 minutes into the future.

While it is difficult to select a good default value that works for all clusters, the following is a general rule of thumb:

- Increase NODEDOWNSTATEDELAYTIME if jobs are getting blocked due to priority reservations sliding as down nodes are not recovered.
- Decrease NODEDOWNSTATEDELAYTIME if high priority job reservations are getting regularly delayed due to transient node failures.

```
# assume down nodes will not be recovered for one hour
NODEDOWNSTATEDELAYTIME  01:00:00
```

4.4.6 Allocated Resource Failure Policy for Jobs

If a failure occurs within a collection of nodes allocated to a job, Moab can automatically re-allocate replacement resources. This can be configured with the parameter JOBACTIONONNODEFAILURE.

How an active job behaves when one or more of its allocated resources fail depends on the allocated resource failure policy. Depending on the type of job, type of resources, and type of middleware infrastructure, you may choose to have different responses based on the job, the resource, and the type of failure.

In this topic:

4.4.6.A Failure Responses

By default, Moab cancels a job when an allocated resource failure is detected. However, you can specify the following actions:

Option	Policy Action
CANCEL	Cancels the job.
FAIL	Terminates the job as a failed job.
HOLD	Places a hold on the job. This option is only applicable if you are using check-pointing.

Option	Policy Action
IGNORE	Ignores the failed node, allowing the job to proceed.
NOTIFY	Notifies the admin and user of failure but takes no further action.
REQUEUE	Requeues job and allows it to run when alternative resources become available.

4.4.6.B Policy Precedence

For a given job, the applied policy can be set at various levels with policy precedence applied in the job, class/queue, partition, and then system level. The following table indicates the available methods for setting this policy:

Object	Parameter	Example
Job	RESFAILPOLICY resource manager extension	`> qsub -l resfailpolicy=requeue`
Class/Queue	RESFAILPOLICY attribute of CLASSCFG parameter	`CLASSCFG[batch] RESFAILPOLICY=CANCEL`
Partition	JOBACTIONONNODEFAILURE attribute of PARCFG parameter	`PARCFG[web3] JOBACTIONONNODEFAILURE=NOTIFY`
System	NODEALLOCRESFAILUREPOLICY parameter	`NODEALLOCRESFAILUREPOLICY=MIGRATE`

4.4.6.C Failure Definition

Any allocated node going down constitutes a failure. However, for certain types of workload, responses to failures may be different depending on whether it is the master task (task 0) or a slave task that fails. To indicate that the associated policy should only take effect if the master task fails, the allocated resource failure policy should be specified with a trailing asterisk (*), as in the following example:

```
CLASSCFG[virtual_services] RESFAILPOLICY=requeue*
```

4.4.6.D Torque Failure Details

When a node fails to send a status update within a configurable time frame (default 600 seconds, see node_check_rate in the *Torque Resource Manager Administrator Guide*), `pbs_server` determines that the node is down. Depending on the `JOBACTIONONNODEFAILURE` parameter setting, Moab may then notify admins, hold the job, requeue the job, allocate replacement resources to the job, or cancel the job. If Moab requests that Torque cancel or requeue the job, Torque immediately frees all non-failed resources, making them available for use by other jobs. `pbs_mom` also cleans up parallel jobs after a configurable time frame (default 600 seconds, see $job_exit_wait_time in the *Torque Resource Manager Administrator Guide*). Once the failed node is recovered, it contacts the resource manager, determines that the associated job has been canceled/requeued, cleans up, and makes itself available for new workload.

Related Topics

- 10.6 Managing Node State
- JOBACTIONONNODEFAILURE parameter
- NODEFAILURERESERVETIME parameter
- NODEDOWNSTATEDELAYTIME parameter (down nodes will be marked unavailable for the specified duration)
- NODEDRAINSTATEDELAYTIME parameter (offline nodes will be marked unavailable for the specified duration)
- NODEBUSYSTATEDELAYTIME parameter (nodes with unexpected background load will be marked unavailable for the specified duration)
- NODEALLOCRESFAILUREPOLICY parameter (action to take if executing jobs have one or more allocated nodes fail)

Chapter 5: Managing Fairness - Throttling Policies, Fairshare, Allocation Management

In this chapter:

5.1 Fairness

The concept of cluster fairness varies widely from person to person and site to site. While some interpret it as giving all users equal access to compute resources, more complicated concepts incorporating historical resource usage, political issues, and job value are equally valid. While no scheduler can address all possible definitions of fair, Moab provides one of the industry's most comprehensive and flexible set of tools allowing most sites the ability to address their many and varied fairness management needs.

In this section:

5.1.1 Fairness Facilities

Under Moab, most fairness policies are addressed by a combination of the facilities described in the following table:

Job Prioritization	
Description	Specifies what is most important to the scheduler. Using service based priority factors enables you to balance job turnaround time, expansion factor, or other scheduling performance metrics.

Job Prioritization	

Example	`SERVICEWEIGHT    1`
	`QUEUETIMEWEIGHT 10`
	Causes jobs to increase in priority by 10 points for every minute they remain in the queue.

Usage Limits (Throttling Policies)	
Description	Specifies limits on exactly what resources can be used at any given instant.

Example	`USERCFG[john]      MAXJOB=3`
	`GROUPCFG[DEFAULT] MAXPROC=64`
	`GROUPCFG[staff]   MAXPROC=128`
	Allows `john` to only run `3` jobs at a time. Allows the group `staff` to use up to `128` total processors and all other groups to use up to `64` processors.

Fairshare	
Description	Specifies usage targets to limit resource access or adjust priority based on historical cluster and grid level resource usage.

Example	`USERCFG[steve]  FSTARGET=25.0+`
	`FSWEIGHT        1`
	`FSUSERWEIGHT    10`
	Enables priority based fairshare and specifies a fairshare target for user `steve` such that his jobs are favored in an attempt to keep his jobs using at least `25.0%` of delivered compute cycles.

Allocation Management	
Description	Specifies long term, credential-based resource usage limits.
Example	`AMCFG[mam]  TYPE=MAM  HOST=server.sys.net` *Enables the Moab Accounting Manager allocation management interface. Within the accounting manager, project or account based allocations can be configured. These allocations can, for example, do such things as allow project X to use up to 100,000 processor-hours per quarter, provide various QoS sensitive charge rates, and share allocation access.*

Quality of Service	
Description	Specifies additional resource and service access for particular users, groups, and accounts. QoS facilities can provide special priorities, policy exemptions, reservation access, and other benefits (and special charge rates).

Example	`QOSCFG[orion] PRIORITY=1000 XFTARGET=1.2`
	`QOSCFG[orion] QFLAGS=PREEMPTOR,IGNSYSTEM,RESERVEALWAYS`
	Enables jobs requesting the `orion` *QoS a priority increase, an expansion factor target to improve response time, the ability to preempt other jobs, an exemption from system level job size policies, and the ability to always reserve needed resources if it cannot start immediately.*

Standing Reservations	
Description	Reserves blocks of resources within the cluster for specific, periodic time frames under the constraints of a flexible access control list.

Example	`SRCFG[jupiter] HOSTLIST=node01[1-4]`
	`SRCFG[jupiter] STARTTIME=9:00:00 ENDTIME=17:00:00`
	`SRCFG[jupiter] USERLIST=john,steve ACCOUNTLIST=jupiter`
	Reserve nodes `node011` *through* `node014` *from 9:00 A.M. until 5:00 P.M. for use by jobs from user* `john` *or* `steve` *or from the project* `jupiter`*.*

Class/Queue Constraints	
Description	Associates users, resources, priorities, and limits with cluster classes or cluster queues that can be assigned to or selected by end-users.

Example	`CLASSCFG[long] HOSTLIST=acn[1-4][0-9]`
	`CLASSCFG[long] MIN.WCLIMIT=24:00:00`
	`SRCFG[jupiter] PRIORITY=10000`
	`SRCFG[jupiter] CLASSLIST=long&`
	Assigns long jobs a high priority but only allows them to run on certain nodes.

5.1.2 Selecting the Correct Policy Approach

Moab supports a rich set of policy controls in some cases allowing a particular policy to be enforced in more than one way. For example, cycle distribution can be controlled using usage limits, fairshare, or even queue definitions. Selecting the most correct policy depends on site objectives and needs; consider the following when making such a decision:

- Minimal end-user training
    - Does the solution use an approach familiar to or easily learned by existing users?
- End-user transparency
    - Can the configuration be enabled or disabled without impacting user behavior or job submission?
- Impact on system utilization and system responsiveness
- Solution complexity
    - Is the impact of the configuration readily intuitive, and is it easy to identify possible side effects?
- Solution extensibility and flexibility
    - Will the proposed approach allow the solution to be easily tuned and extended as cluster needs evolve?

Related Topics

- 4.1 Job Prioritization
- 5.2 Usage Limits/Throttling Policies
- 5.3 Fairshare
- 5.5 Accounting, Charging, and Allocation Management
- 6.3 Quality of Service (QoS) Facilities
- 6.1.3 Standing Reservations
- 2.7.5 Class (or Queue) Credential - Class (or Queue) Credential constraints

5.2 Usage Limits/Throttling Policies

A number of Moab policies enable an admin to control job flow through the system. These throttling policies work as filters allowing or disallowing a job to be considered for scheduling by specifying limits regarding system usage for any given moment. These policies can be specified as global or specific constraints specified on a per user, group, account, QoS, or class basis.

In this section:

5.2.1 Fairness via Throttling Policies

Moab allows significant flexibility with usage limits, or throttling policies. At a high level, Moab allows resource usage limits to be specified in three primary workload categories: (1) active, (2) idle, and (3) system job limits.

In this topic:

5.2.1.A Basic Fairness Policies

Workload Category	Description
Active job limits	Constrain the total cumulative resources available to active jobs at a given time.
Idle job limits	Constrain the total cumulative resources available to idle jobs at a given time.
System job limits	Constrain the maximum resource requirements of any single job.

These limits can be applied to any job credential (user, group, account, QoS, and class), or on a system-wide basis. Using the keyword `DEFAULT`, you can also specify the default setting for the desired user, group, account, QoS, and class. Additionally, you can configure QoS to allow limit overrides to any particular policy.

To run, a job must meet all policy limits. Limits are applied using the `*CFG` set of parameters, particularly USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, CLASSCFG, and

SYSCFG. Limits are specified by associating the desired limit to the individual or default object. The usage limits currently supported are listed below:

Usage Limits

MAXARRAYJOB	
Units	Number of simultaneous active array job subjobs.
Description	Limits the number of simultaneously active (starting or running) array subjobs a credential can have.
Example	`USERCFG[gail] MAXARRAYJOB=10` *Gail can have a maximum of 10 active job array subjobs.*

MAXGRES	
Units	# of concurrent uses of a generic resource
Description	Limits the concurrent usage of a generic resource to a specific quantity or quantity range.

Example	`USERCFG[joe] MAXGRES[matlab]=2`
	 `USERCFG[jim] MAXGRES[matlab]=2,4`

MAXJOB	
Units	# of jobs
Description	Limits the number of jobs a credential can have active (starting or running) at any given time. Moab places a hold on all new jobs submitted by that credential once it has reached its maximum number of allowable jobs. `MAXJOB=0` is not supported. You can, however, achieve similar results by using the HOLD attribute of the USERCFG parameter: `USERCFG[john] HOLD=yes`

Example	`USERCFG[DEFAULT] MAXJOB=8`
	 `GROUPCFG[staff]  MAXJOB=2,4`

MAXMEM	
Units	total memory in MB
Description	Limits the total amount of dedicated memory (in MB) that can be allocated by a credential's active jobs at any given time.
Example	`ACCOUNTCFG[jasper] MAXMEM=2048`

MAXNODE	
Units	# of nodes
Description	Limits the total number of compute nodes that can be in use by active jobs at any given time. We recommend that you set JOBNODEMATCHPOLICY EXACTNODE when using **MAXNODE**. This ensures jobs submitted using the msub/qsub "-l nodes=#" syntax will have a node count associated with the request. On some systems (including Torque/PBS), nodes have been softly defined rather than strictly defined; that is, a job may request 2 nodes but Torque will translate this request into 1 node with 2 processors. This can prevent Moab from enforcing a **MAXNODE** policy correctly for a single job. Correct behavior can be achieved using **MAXPROC**.
Example	`CLASSCFG[batch] MAXNODE=64`

MAXPE	
Units	# of processor equivalents
Description	Limits the total number of dedicated processor-equivalents that can be allocated by active jobs at any given time.
Example	`QOSCFG[base] MAXPE=128`

MAXPROC	
Units	# of processors
Description	Limits the total number of dedicated processors that can be allocated by active jobs at any given time per credential. To set `MAXPROC` per job, use msub -W.
Example	`CLASSCFG[debug] MAXPROC=32`

MAXPS	
Units	<# of processors> * <walltime>
Description	Limits the number of outstanding processor-seconds a credential can have allocated at any given time. For example, if a user has a 4-processor job that will complete in 1 hour and a 2-processor job that will complete in 6 hours, they have 4 * 1 * 3600 + 2 * 6 * 3600 = 16 * 3600 outstanding processor-seconds. The outstanding processor-second usage of each credential is updated each scheduling iteration, decreasing as jobs approach their completion time.
Example	`USERCFG[DEFAULT] MAXPS=720000`

MAXSUBMITJOBS	
Units	# of jobs
Description	Limits the number of jobs a credential can submit and have in the system at once. Moab will reject any job submitted beyond this limit. If you use a Torque resource manager, you should also set `max_user_queuable` in case the user submits jobs via *`qsub`* instead of *`msub`*. See 'Queue Attributes' in the *Torque Resource Manager Administrator Guide* for more information.
Example	`USERCFG[DEFAULT] MAXSUBMITJOBS=5`

MAXWC	
Units	job duration [[[DD:]HH:]MM:]SS
Description	Limits the cumulative remaining walltime a credential can have associated with active jobs. It behaves identically to the MAXPS limit (listed earlier) only lacking the processor weighting. Like MAXPS, the cumulative remaining walltime of each credential is also updated each scheduling iteration. MAXWC does not limit the maximum wallclock limit per job. For this capability, use the parameter MAX.WCLIMIT.
Example	`USERCFG[ops] MAXWC=72:00:00`

The following example demonstrates a simple limit specification:

```
USERCFG[DEFAULT]  MAXJOB=4
USERCFG[john]     MAXJOB=8
```

This example allows user `john` to run up to `8` jobs while all other users can only run up to `4`.

Simultaneous limits of different types can be applied per credential and multiple types of credentials can have limits specified. This example demonstrates this mixing of limits and is a bit more complicated:

```
USERCFG[steve]    MAXJOB=2 MAXNODE=30
GROUPCFG[staff]   MAXJOB=5
CLASSCFG[DEFAULT] MAXNODE=16
CLASSCFG[batch]   MAXNODE=32
```

This configuration may potentially apply multiple limits to a single job. As discussed previously, a job can only run if it satisfies all applicable limits. Therefore, in this example, the scheduler will be constrained to allow at most `2` simultaneous user `steve` jobs with an aggregate node consumption of no more than `30` nodes. However, if the job is submitted to a class other than `batch`, it may be limited further. Here, only `16` total nodes can be used simultaneously by jobs running in any given class with the exception of the class `batch`. If `steve` submitted a job to run in the class `interactive`, for example, and there were jobs already running in this class using a total of 14 nodes, his job would be blocked unless it requested 2 or fewer nodes by the default limit of `16` nodes per class.

5.2.1.B Multi-Dimension Fairness Policies and Per Credential Overrides

Multi-dimensional fairness policies allow you to specify policies based on combinations of job credentials. A common example might be setting a maximum number of jobs allowed per queue per user or a total number of processors per group per QoS. As with basic fairness policies, multi-dimension policies are specified using the `*CFG` parameters or through the identity manager interface. Moab supports the most commonly used multi-dimensional fairness policies (listed in the Multi-dimensional Usage Limit Permutations table below) using the following format:

`*CFG[X] <LIMITTYPE>[<CRED>]=<LIMITVALUE>`

`*CFG` is one of `USERCFG`, `GROUPCFG`, `ACCOUNTCFG`, `QOSCFG`, or `CLASSCFG`, the `<LIMITTYPE>` policy is one of the policies listed in the table in section 6.2.1.1, and `<CRED>` is of the format `<CREDTYPE>[:<VALUE>]` with `CREDTYPE` being one of `USER`, `GROUP`, `ACCT`, `QoS`, or `CLASS`. The optional `<VALUE>` setting can be used to specify that the policy only applies to a specific credential value. For example, the following configuration sets limits on the class `fast`, controlling the maximum number of jobs any group can have active at any given time and the number of processors in use at any given time for user `steve`.

```
CLASSCFG[fast] MAXJOB[GROUP]=12
CLASSCFG[fast] MAXPROC[USER:steve]=50
CLASSCFG[fast] MAXIJOB[USER]=10
```

The following example configuration may clarify further:

```
# allow class batch to run up the 3 simultaneous jobs
# allow any user to use up to 8 total nodes within class
CLASSCFG[batch] MAXJOB=3 MAXNODE[USER]=8
# allow users steve and bob to use up to 3 and 4 total processors respectively within
class
CLASSCFG[fast] MAXPROC[USER:steve]=3 MAXPROC[USER:bob]=4
```

Multi-dimensional policies cannot be applied on `DEFAULT` credentials.

The table below lists the currently implemented, multi-dimensional usage limit permutations. The slmt stands for Soft Limit, and hlmt stands for Hard Limit.

Multi-dimensional Usage Limit Permutations

Multi-dimension usage limit permutations	
ACCOUNTCFG[name]	MAXIJOB[QOS]=hlmt MAXIJOB[QOS:qosname]=hlmt
	MAXIPROC[QOS]=hlmt MAXIPROC[QOS:qosname]=hlmt
	MAXJOB[QOS]=slmt,hlmt MAXJOB[QOS:qosname]=slmt,hlmt
	MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=slmt,hlmt
	MAXMEM[USER]=slmt,hlmt MAXMEM[USER:username]=slmt,hlmt
	MAXNODE[USER]=slmt,hlmt MAXNODE[USER:username]=slmt,hlmt
	MAXPE[QOS]=slmt,hlmt MAXPE[QOS:qosname]=slmt,hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPROC[QOS]=slmt,hlmt MAXPROC[QOS:qosname]=slmt,hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPS[QOS]=slmt,hlmt MAXPS[QOS:qosname]=slmt,hlmt
	MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=slmt,hlmt

Multi-dimension usage limit permutations	
	MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=slmt,hlmt

CLASSCFG[name]	MAXIJOB[USER]=hlmt
	MAXJOB[GROUP]=slmt,hlmt
	MAXJOB[GROUP:groupname]=slmt,hlmt

	MAXJOB[QOS:qosname]=hlmt
	MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=slmt,hlmt
	MAXMEM[GROUP]=slmt,hlmt MAXMEM[GROUP]=slmt,hlmt
	MAXMEM[GROUP]=slmt,hlmt MAXMEM[GROUP:groupname]=slmt,hlmt
	MAXMEM[QOS:qosname]=hlmt
	MAXMEM[USER]=slmt,hlmt MAXMEM[USER:username]=slmt,hlmt
	MAXNODE[GROUP]=slmt,hlmt MAXNODE[GROUP:groupname]=slmt,hlmt
	MAXNODE[QOS:qosname]=hlmt
	MAXNODE[USER]=slmt,hlmt MAXNODE[USER:username]=slmt,hlmt
	MAXPE[GROUP]=slmt,hlmt MAXPE[GROUP:groupname]=slmt,hlmt
	MAXPE[QOS:qosname]=hlmt
	MAXPE[USER]=slmt,hlmt

Multi-dimension usage limit permutations	
	MAXPE[USER:username]=slmt,hlmt
	MAXPROC[GROUP]=slmt,hlmt MAXPROC[GROUP:groupname]=slmt,hlmt
	MAXPROC[QOS:qosname]=hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPS[GROUP]=slmt,hlmt MAXPS[GROUP:groupname]=slmt,hlmt
	MAXPS[QOS:qosname]=hlmt
	MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=slmt,hlmt
	MAXWC[GROUP]=slmt,hlmt MAXWC[GROUP:groupname]=slmt,hlmt
	MAXWC[QOS:qosname]=hlmt
	MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=slmt,hlmt

Multi-dimension usage limit permutations	
GROUPCFG[name]	MAXJOB[CLASS:classname]=slmt,hlmt
	MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=slmt,hlmt
	MAXMEM[CLASS:classname]=slmt,hlmt
	MAXMEM[USER]=slmt,hlmt MAXMEM[USER:username]=slmt,hlmt
	MAXNODE[CLASS:classname]=slmt,hlmt
	MAXNODE[USER]=slmt,hlmt MAXNODE[USER:username]=slmt,hlmt
	MAXPE[CLASS:classname]=slmt,hlmt
	MAXPE[USER]=slmt,hlmt MAXPE[USER:username]=slmt,hlmt
	MAXPROC[CLASS:classname]=slmt,hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPS[CLASS:classname]=slmt,hlmt
	MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=slmt,hlmt
	MAXWC[CLASS:classname]=slmt,hlmt
	MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=slmt,hlmt
QOSCFG[name]	MAXIJOB[ACCT]=hlmt MAXIJOB[ACCT:accountname]=hlmt

Multi-dimension usage limit permutations	
	MAXIJOB[USER]=hlmt MAXIJOB[USER:class+classname]=hlmt
	MAXINODE[ACCT]=slmt,hlmt MAXINODE[ACCT:accountname]=slmt,hlmt
	MAXINODE[USER]=hlmt MAXINODE[USER:username]=slmt,hlmt
	MAXIPROC[ACCT]=hlmt MAXIPROC[ACCT:accountname]=hlmt
	MAXJOB[ACCT]=slmt,hlmt MAXJOB[ACCT:accountname]=slmt,hlmt
	MAXJOB[USER]=slmt,hlmt MAXJOB[USER:username]=slmt,hlmt
	MAXMEM[USER]=slmt,hlmt MAXMEM[USER:username]=slmt,hlmt
	MAXNODE[USER]=slmt,hlmt MAXNODE[USER:username]=slmt,hlmt
	MAXPE[ACCT]=slmt,hlmt MAXPE[ACCT:accountname]=slmt,hlmt
	MAXPE[USER]=slmt,hlmt MAXPE[USER:username]=slmt,hlmt
	MAXPROC[ACCT]=slmt,hlmt MAXPROC[ACCT:accountname]=slmt,hlmt
	MAXPROC[USER]=slmt,hlmt MAXPROC[USER:username]=slmt,hlmt
	MAXPS[ACCT]=slmt,hlmt

Multi-dimension usage limit permutations	
	MAXPS[ACCT:accountname]=slmt,hlmt
	MAXPS[USER]=slmt,hlmt MAXPS[USER:username]=slmt,hlmt
	MAXWC[USER]=slmt,hlmt MAXWC[USER:username]=slmt,hlmt
USERCFG[name]	MAXJOB[GROUP]=slmt,hlmt MAXJOB[GROUP:groupname]=slmt,hlmt
	MAXMEM[GROUP]=slmt,hlmt MAXMEM[GROUP:groupname]=slmt,hlmt
	MAXNODE[GROUP]=slmt,hlmt MAXNODE[GROUP:groupname]=slmt,hlmt
	MAXPE[GROUP]=slmt,hlmt MAXPE[GROUP:groupname]=slmt,hlmt
	MAXPROC[GROUP]=slmt,hlmt MAXPROC[GROUP:groupname]=slmt,hlmt
	MAXPS[GROUP]=slmt,hlmt MAXPS[GROUP:groupname]=slmt,hlmt
	MAXWC[GROUP]=slmt,hlmt MAXWC[GROUP:groupname]=slmt,hlmt

5.2.2 Override Limits

Like all job credentials, the QoS object can be associated with resource usage limits. However, this credential can also be given special override limits that supersede the limits of other credentials, effectively causing all other limits of the same type to be ignored. See 6.3.2.C Usage Limits and Overrides for a complete list of policies that can be overridden. The following configuration provides an example of this in the last line:

```
USERCFG[steve]    MAXJOB=2   MAXNODE=30
GROUPCFG[staff]   MAXJOB=5
```

```
CLASSCFG[DEFAULT] MAXNODE=16
CLASSCFG[batch]   MAXNODE=32
QOSCFG[hiprio]    OMAXJOB=3  OMAXNODE=64
```

Only 3 `hiprio` QoS jobs can run simultaneously and `hiprio` QoS jobs can run with up to 64 nodes per credential ignoring other credential `MAXNODE` limits.

Given the preceding configuration, assume a job is submitted with the credentials, user `steve`, group `staff`, class `batch`, and QoS `hiprio`.

Such a job will start so long as running it does not lead to any of the following conditions:

- Total nodes used by user `steve` does not exceed `64`.
- Total active jobs associated with user `steve` does not exceed `2`.
- Total active jobs associated with group `staff` does not exceed `5`.
- Total nodes dedicated to class `batch` does not exceed `64`.
- Total active jobs associated with QoS `hiprio` does not exceed `3`.

While the preceding example is a bit complicated for most sites, similar combinations may be required to enforce policies found on many systems.

5.2.3 Idle Job Limits

Idle (or queued) job limits control which jobs are eligible for scheduling. To be eligible for scheduling, a job must meet the following conditions:

- Be idle as far as the resource manager is concerned (no holds).
- Have all job prerequisites satisfied (no outstanding job or data dependencies).
- Meet all idle job throttling policies.

If a job fails to meet any of these conditions, it will not be considered for scheduling and will not accrue service based job prioritization (see 4.1.2.E Service (SERVICE) Component and the JOBPRIOACCRUALPOLICY parameter) The primary purpose of idle job limits is to ensure fairness among competing users by preventing queue stuffing and other similar abuses. Queue stuffing occurs when a single entity submits large numbers of jobs, perhaps thousands, all at once so they begin accruing queue time based priority and remain first to run despite subsequent submissions by other users.

Idle limits are specified in a manner almost identical to active job limits with the insertion of the capital letter *I* into the middle of the limit name. The following tables describe the `MAXIARRAYJOB`, `MAXIJOB`, and `MAXINODE` limits, which are idle limit equivalents to MAXARRAYJOB, MAXJOB, and MAXNODE limits, respectively.

MAXIARRAYJOB	
Units	Number of simultaneous idle array job subjobs.
Description	Limits the number of simultaneously idle (eligible) job array subjobs across *all* job arrays submitted by a credential.
Example	`USERCFG[gail] MAXARRAYJOB=10 MAXIARRAYJOB=5` *Gail can have a maximum of 10 active job array subjobs and 5 eligible job array subjobs.*

MAXIJOB	
Units	# of jobs
Description	Limits the number of idle (eligible) jobs a credential can have at any given time.

Example	`USERCFG[DEFAULT]  MAXIJOB=8`
	`GROUPCFG[staff]   MAXIJOB=4`

MAXINODE	
Units	# of nodes
Description	Limits the total number of compute nodes that can be requested by jobs in the eligible/idle queue at any time. Once the limit is exceeded, the remaining jobs will be placed in the blocked queue. The number of nodes is determined by `<tasks>` / `<maximumProcsOnOneNode>` or, if using JOBNODEMATCHPOLICY`EXACTNODE`, by the number of nodes requested.
Example	`USERCFG[DEFAULT]  MAXINODE=2`

Idle limits can constrain the total number of jobs considered to be eligible on a per credential basis. Further, like active job limits, idle job limits can also constrain eligible jobs based on aggregate requested resources. This could, for example, allow you to indicate that for a given user, only jobs requesting up to a total of 64 processors, or 3200 processor-seconds are considered at any given time. Which jobs to select is accomplished by prioritizing all idle jobs and then adding jobs to the eligible list one at a time in priority order until jobs can no longer be added. This eligible job selection is done only once per scheduling iteration, so, consequently, idle job limits only support a single hard limit specification. Any specified soft limit is ignored.

All single dimensional job limit types supported as active job limits are also supported as idle job limits. In addition, Moab also supports MAXIJOB[USER] and MAXIPROC[USER] policies on a per class basis (see the section Basic Fairness Policies above).

Example

```
USERCFG[steve]     MAXIJOB=2
GROUPCFG[staff]    MAXIJOB=5
CLASSCFG[batch]    MAXIJOB[USER]=2 MAXIJOB[USER:john]=6
QOSCFG[hiprio]     MAXIJOB=3
```

5.2.4 Hard and Soft Limits

Hard and soft limit specification enables you to balance both fairness and utilization on a given system. Typically, throttling limits are used to constrain the quantity of resources a given credential (such as user or group) is allowed to consume. These limits can be very effective in enforcing fair usage among a group of users. However, in a lightly loaded system, or one in which there are significant swings in usage from project to project, these limits can reduce system utilization by blocking jobs even when no competing jobs are queued.

Soft limits help address this problem by providing additional scheduling flexibility. They allow sites to specify two tiers of limits; the more constraining limits soft limits are in effect in heavily loaded situations and reflect tight fairness constraints. The more flexible hard limits specify how flexible the scheduler can be in selecting jobs when there are idle resources available after all jobs meeting the tighter soft limits have started. Soft and hard limits are specified in the format `[<SOFTLIMIT>,]<HARDLIMIT>`. For example, a given site may want to use the following configuration:

```
USERCFG[DEFAULT]  MAXJOB=2,8
```

With this configuration, the scheduler selects all jobs that meet the per user `MAXJOB` limit of `2`. It then attempts to start and reserve resources for all of these selected jobs. If after doing so, there still remain available resources, the scheduler then selects all jobs that meet the less constraining hard per user `MAXJOB` limit of `8` jobs. These jobs would then be scheduled and reserved as available resources allow.

If no soft limit is specified or the soft limit is less constraining than the hard limit, the soft limit is set equal to the hard limit.

Example:

```
USERCFG[steve]     MAXJOB=2,4 MAXNODE=15,30
GROUPCFG[staff]    MAXJOB=2,5
CLASSCFG[DEFAULT]  MAXNODE=16,32
CLASSCFG[batch]    MAXNODE=12,32
QOSCFG[hiprio]     MAXJOB=3,5 MAXNODE=32,64
```

Job preemption status can be adjusted based on whether the job violates a soft policy using the ENABLESPVIOLATIONPREEMPTION parameter.

5.2.5 Per-partition Limits

Per-partition scheduling can set limits and enforce credentials and polices on a per-partition basis. Configuration for per-partition scheduling is done on the Moab Grid Control. In a grid, each Moab cluster is considered a partition. Per-partition scheduling is typically used in a Moab Grid Control / Moab Grid Member grid.

To enable per-partition scheduling, add the following to `moab.cfg`:

```
PERPARTITIONSCHEDULING TRUE
JOBMIGRATEPOLICY JUSTINTIME
```

With per-partition scheduling, we recommend that limits go on the specific partitions and not on the global level. If limits are specified on both levels, Moab will take the more constricting of the limits. Also, note that a DEFAULT policy on the global partition is not overridden by any policy on a specific partition.

In this topic:

5.2.5.A Configuring Per-partition Limits

You can configure per-job limits and credential usage limits on a per-partition basis in the `moab.cfg` file. Here is a sample configuration for partitions `g02` and `g03` in `moab.cfg`:

```
PARCFG[g02]   CONFIGFILE=/opt/moab/parg02.cfg
PARCFG[g03]   CONFIGFILE=/opt/moab/parg03.cfg
```

You can then add per-partition limits in each partition configuration file:

```
# /opt/moab/parg02.cfg
CLASSCFG[pbatch]   MAXJOB=5
```

```
# /opt/moab/parg03.cfg
CLASSCFG[pbatch]   MAXJOB=10
```

You can configure Moab so that jobs submitted to any partition besides `g02`and `g03` get the default limits in `moab.cfg`:

```
stl

CLASSCFG[pbatch]  MAXJOB=2
```

5.2.5.B Supported Credentials and Limits

The user, group, account, QoS, and class credentials are supported in per-partition scheduling.

The following per-job limits are supported:

- MAX.NODE
- MAX.WCLIMIT
- MAX.PROC

The following credential usage limits are supported:

- MAXJOB
- MAXNODE
- MAXPROC
- MAXWC
- MAXSUBMITJOBS

Multi-dimensional limits are supported for the listed credentials and per-job limits. For example:

```
CLASSCFG[pbatch]   MAXJOB[user:frank]=10
```

5.2.6 Usage-Based Limits

Resource usage limits constrain the amount of resources a given job can consume. These limits are generally proportional to the resources requested and can include walltime, any standard resource, or any specified generic resource. The parameter RESOURCELIMITPOLICY controls which resources are limited, what limit policy is enforced per resource, and what actions the scheduler should take in the event of a policy violation.

In this topic:

5.2.6.A Configuring Actions

The `RESOURCELIMITPOLICY` parameter accepts a number of policies, resources, and actions using the format and values defined below.

If walltime is the resource to be limited, be sure that the resource manager is configured to not interfere if a job surpasses its given walltime. For Torque, this is done by using $ignwalltime in the configuration on each MOM (Machine Oriented Mini-server) node.

5.2.6.B Format

```
RESOURCELIMITPOLICY<RESOURCE>:[<SPOLICY>,]<HPOLICY>:
[<SACTION>,]<HACTION>[:[<SVIOLATIONTIME>,]<HVIOLATIONTIME>]...
```

Resource	Description
CPUTIME	Maximum total job proc-seconds used by any single job (allows scheduler enforcement of cpulimit).
DISK	Local disk space (in MB) used by any single job task.
JOBMEM	Maximum real memory/RAM (in MB) used by any single job. JOBMEM will only work with the MAXMEM flag.
JOBPROC	Maximum processor load associated with any single job. You must set MAXPROC to use `JOBPROC`.
MEM	Maximum real memory/RAM (in MB) used by any single job task.
MINJOBPROC	Minimum processor load associated with any single job (action taken if job is using 5% or less of potential CPU usage).
NETWORK	Maximum network load associated with any single job task.
PROC	Maximum processor load associated with any single job task.
SWAP	Maximum virtual memory/SWAP (in MB) used by any single job task.
WALLTIME	Requested job walltime.

Policy	Description
ALWAYS	Take action whenever a violation is detected.
EXTENDEDVIOLATION	Take action only if a violation is detected and persists for greater than the specified time limit.
BLOCKEDWORKLOADONLY	Take action only if a violation is detected and the constrained resource is required by another job.

Action	Description
CANCEL	Terminate the job.
CHECKPOINT	checkpoint and terminate job.
MIGRATE	Requeue the job and require a different set of hosts for execution.
NOTIFY	Notify admins and job owner regarding violation.
REQUEUE	Terminate and requeue the job.
SUSPEND	Suspend the job and leave it suspended for an amount of time defined by the MINADMINSTIME parameter.

Example 5-1: Notify and then cancel job if requested memory is exceeded

```
# if job exceeds memory usage, immediately notify owner
# if job exceeds memory usage for more than 5 minutes, cancel the job
RESOURCELIMITPOLICY MEM:ALWAYS,EXTENDEDVIOLATION:NOTIFY,CANCEL:00:05:00
```

Example 5-2: Checkpoint job on walltime violations

```
# if job exceeds requested walltime, checkpoint job
RESOURCELIMITPOLICY WALLTIME:ALWAYS:CHECKPOINT
# when checkpointing, send term signal, followed by kill 1 minute later
RMCFG[base] TYPE=PBS CHECKPOINTTIMEOUT=00:01:00 CHECKPOINTSIG=SIGTERM
```

Example 5-3: Cancel jobs that use 5% or less of potential CPU usage for more than 5 minutes

```
RESOURCELIMITPOLICY MINJOBPROC:EXTENDEDVIOLATION:CANCEL:5:00
```

Example 5-4: Migrating a job when it blocks other workload

```
RESOURCELIMITPOLICY JOBPROC:BLOCKEDWORKLOADONLY:MIGRATE
```

5.2.6.C Specifying Hard and Soft Policy Violations

Moab is able to perform different actions for both hard and soft policy violations. In most resource management systems, a mechanism does not exist to allow the user to specify both hard and soft limits. To address this, Moab provides the RESOURCELIMITMULTIPLIER parameter that allows per partition and per resource multiplier factors to be specified to generate the actual hard and soft limits to be used. If the factor is less than one, the soft limit will be lower than the specified value and a Moab action will be taken before the specified limit is reached. If the factor is greater than one, the hard limit will be set higher than the specified limit allowing a buffer space before the hard limit action is taken.

In the following example, job owners will be notified by email when their memory reaches 100% of the target, and the job will be canceled if it reaches 125% of the target. For wallclock usage, the job will be requeued when it reaches 90% of the specified limit if another job is waiting for its resources, and it will be checkpointed when it reaches the full limit.

```
RESOURCELIMITPOLICY       MEM:ALWAYS,ALWAYS:NOTIFY,CANCEL
RESOURCELIMITPOLICY       WALLTIME:BLOCKEDWORKLOADONLY,ALWAYS:REQUEUE,CHECKPOINT
RESOURCELIMITMULTIPLIER   MEM:1.25,WALLTIME:0.9
```

5.2.6.D Constraining Walltime Usage

While Moab constrains walltime using the parameter RESOURCELIMITPOLICY like other resources, it also allows walltime exception policies that are not available with other resources. In particular, Moab allows jobs to exceed the requested wallclock limit by an amount specified on a global basis using the JOBMAXOVERRUN parameter or on a per credential basis using the `WCOVERRUN` attribute of the CLASSCFG parameter.

```
JOBMAXOVERRUN    00:10:00
CLASSCFG[debug]  wcoverrun=00:00:30
```

Related Topics

- RESOURCELIMITPOLICY parameter
- FSTREE parameter (set usage limits within share tree hierarchy)
- 2.7 Credentials
- JOBMAXOVERRUN parameter
- WCVIOLATIONACTION parameter
- RESOURCELIMITMULTIPLIER parameter

5.3 Fairshare

Fairshare allows historical resource utilization information to be incorporated into job feasibility and priority decisions. This feature allows site admins to set system utilization targets for users, groups, accounts, classes, and QoS levels. Admins can also specify the time frame over which resource utilization is evaluated in determining whether the goal is being reached. Parameters allow sites to specify the utilization metric, how historical information is aggregated, and the effect of fairshare state on scheduling behavior. You can specify fairshare targets for any credentials (such as user, group, and class) that admins want such information to affect.

In this section:

5.3.1 Fairshare Parameters

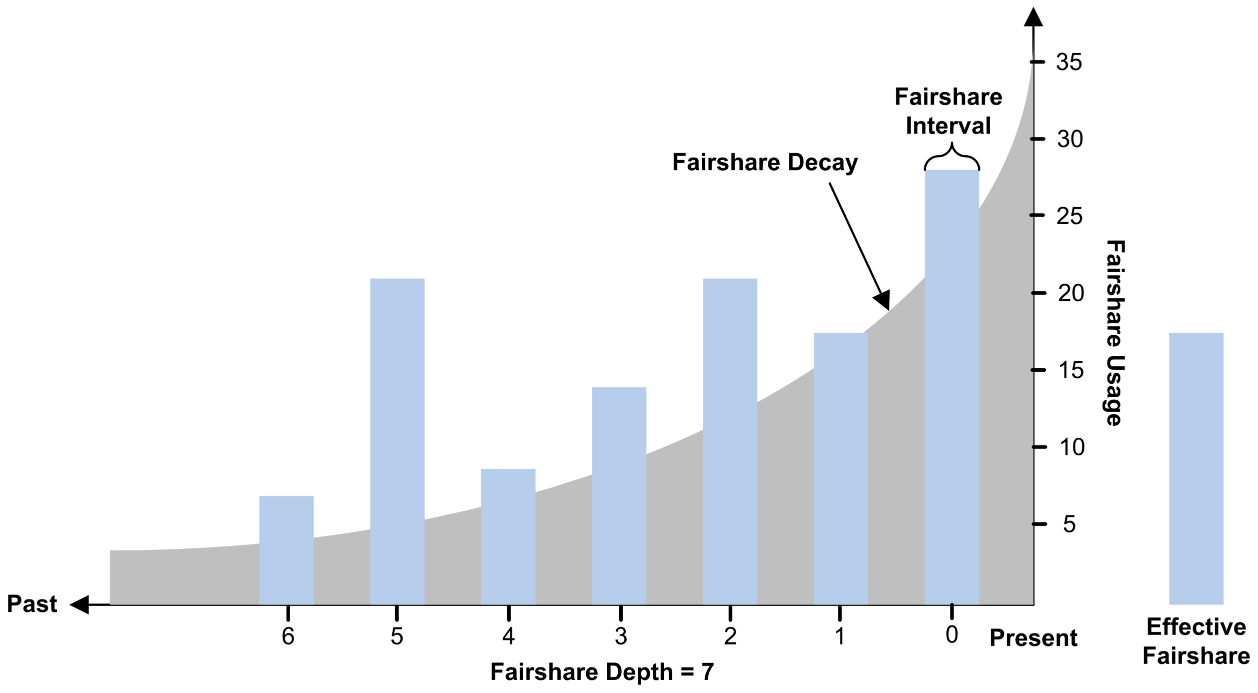
Fairshare is configured at two levels. First, at a system level, configuration is required to determine how fairshare usage information is to be collected and processed. Second, some configuration is required at the credential level to determine how this fairshare information affects particular jobs. The following are system level parameters:

Parameter	Description
FSINTERVAL	Duration of each fairshare window.
FSDEPTH	Number of fairshare windows factored into current fairshare utilization.
FSDECAY	Decay factor applied to weighting the contribution of each fairshare window.
FSPOLICY	Metric to use when tracking fairshare usage.

Credential level configuration consists of specifying fairshare utilization targets using the `*CFG` suite of parameters, including ACCOUNTCFG, CLASSCFG, GROUPCFG, QOSCFG, and USERCFG.

If global (multi-cluster) fairshare is used, Moab must be configured to synchronize this information with an identity manager.

Image 5-1: Effective fairshare over 7 days



5.3.1.A FSPOLICY - Specifying the Metric of Consumption

As Moab runs, it records how available resources are used. Each iteration (`RMPOLLINTERVAL` seconds), it updates fairshare resource utilization statistics. Resource utilization is tracked in accordance with the FSPOLICY parameter allowing various aspects of resource consumption information to be measured. This parameter allows selection of both the types of resources to be tracked and the method of tracking. It provides the option of tracking usage by dedicated or consumed resources, where dedicated usage tracks what the scheduler assigns to the job and consumed usage tracks what the job actually uses.

Metric	Description
DEDICATEDPES	Usage tracked by processor-equivalent seconds dedicated to each job. This is based on the total number of dedicated processor-equivalent seconds delivered in the system. Useful in dedicated and shared nodes environments.

Metric	Description
DEDICATEDPS	Usage tracked by processor seconds dedicated to each job. This is based on the total number of dedicated processor seconds delivered in the system. Useful in dedicated node environments.
DEDICATEDPS%	Usage tracked by processor seconds dedicated to each job. This is based on the total number of dedicated processor seconds *available* in the system.
[NONE]	Disables fairshare.
UTILIZEDPS	Usage tracked by processor seconds used by each job. This is based on the total number of utilized processor seconds delivered in the system. Useful in shared node/SMP environments.

Example 5-5: FSPOLICY

An example may clarify the use of the FSPOLICY parameter. Assume a 4-processor job is running a parallel /bin/sleep for 15 minutes. It will have a dedicated fairshare usage of 1 processor-hour but a consumed fairshare usage of essentially nothing since it did not consume anything. Most often, dedicated fairshare usage is used on dedicated resource platforms while consumed tracking is used in shared SMP environments.

```
FSPOLICY    DEDICATEDPS%
FSINTERVAL  24:00:00
FSDEPTH     28
FSDECAY     0.75
```

5.3.1.B Specifying Fairshare Timeframe

When configuring fairshare, it is important to determine the proper timeframe that should be considered. Many sites choose to incorporate historical usage information from the last one to two weeks while others are only concerned about the events of the last few hours. The correct setting is very site dependent and usually incorporates both average job turnaround time and site mission policies.

With Moab's fairshare system, time is broken into a number of distinct fairshare windows. Sites configure the amount of time they want to consider by specifying two parameters, FSINTERVAL and FSDEPTH. The FSINTERVAL parameter specifies the duration of each window while the FSDEPTH parameter indicates the number of windows to consider. Therefore, the total time evaluated by fairshare is simply FSINTERVAL * FSDEPTH.

Many sites want to limit the impact of fairshare data according to its age. The FSDECAY parameter allows this, causing the most recent fairshare data to contribute more to a credential's total fairshare usage than older data. This parameter is specified as a standard decay factor, which is applied to the fairshare data. Generally, decay factors are specified as a value between 1 and 0 where a value of `1` (the default) indicates no decay should be specified. The smaller the number, the more rapid the decay using the calculation `WeightedValue = Value * <DECAY> ^ <N>`, where `<N>` is the window number. The following table shows the impact of a number of commonly used decay factors on the percentage contribution of each fairshare window:

Decay Factor	Win0	Win1	Win2	Win3	Win4	Win5	Win6	Win7
1.00	100%	100%	100%	100%	100%	100%	100%	100%
0.80	100%	80%	64%	51%	41%	33%	26%	21%
0.75	100%	75%	56%	42%	31%	23%	17%	12%
0.50	100%	50%	25%	13%	6%	3%	2%	1%

While selecting how the total fairshare time frame is broken up between the number and length of windows is a matter of preference, it is important to note that more windows will cause the decay factor to degrade the contribution of aged data more quickly.

5.3.1.C Managing Fairshare Data

Using the selected fairshare usage metric, Moab continues to update the current fairshare window until it reaches a fairshare window boundary, at which point it rolls the fairshare window and begins updating the new window. The information for each window is stored in its own file located in the Moab statistics directory. Each file is named `FS.<EPOCHTIME>[.<PNAME>]`, where `<EPOCHTIME>` is the time the new fairshare window became active (see 5.4 Sample FairShare Data File) and `<PNAME>` is only used if per-partition share trees are configured. Each window contains utilization information for each entity, and for total usage.

Historical fairshare data is recorded in the fairshare file using the metric specified by the FSPOLICY parameter. By default, this metric is processor-seconds.

Historical fairshare data can be directly analyzed and reported using the mdiag -f -v command.

When Moab needs to determine current fairshare usage for a particular credential, it calculates a decay-weighted average of the usage information for that credential using the most recent fairshare intervals where the number of windows evaluated is controlled by the FSDEPTH parameter. For example, assume the credential of interest is user `john` and the following parameters are set:

```
FSINTERVAL    12:00:00
FSDEPTH       4
FSDECAY       0.5
```

Further assume that the fairshare usage intervals have the following usage amounts:

Fairshare interval	Total user `john` usage	Total cluster usage
0	60	110
1	0	125
2	10	100
3	50	150

Based on this information, the current fairshare usage for user `john` is calculated as this:

*Usage = (60 * 1 + .5^1 * 0 + .5^2 * 10 + .5^3 * 50) / (110 + .5^1*125 + .5^2*100 + .5^3*150)*

The current fairshare usage is relative to the actual resources delivered by the system over the timeframe evaluated, not the resources available or configured during that time.

Historical fairshare data is organized into a number of data files, each file containing the information for a length of time as specified by the FSINTERVAL parameter. Although `FSDEPTH`, `FSINTERVAL`, and `FSDECAY` can be freely and dynamically modified, such changes might result in unexpected fairshare status for a period of time as the fairshare data files with the old `FSINTERVAL` setting are rolled out.

5.3.2 Using Fairshare Information

In this topic:

5.3.2.A Fairshare Targets

Once the global fairshare policies have been configured, the next step involves applying resulting fairshare usage information to affect scheduling behavior. As mentioned in the Fairshare overview, by specifying fairshare targets, site admins can configure how fairshare information impacts scheduling behavior. The targets can be applied to user, group, account, QoS, or class credentials using the `FSTARGET` attribute of `*CFG` credential parameters. These targets allow fairshare information to affect job priority and each target can be independently selected to be one of the types documented in the following table:

Target type - Ceiling	
Target modifier	-
Job impact	Priority
Format	Percentage Usage
Description	Adjusts job priority down when usage exceeds target. See the section How Violated Ceilings and Floors Affect Fairshare-Based Priority for more information on how ceilings affect job priority.

Target type - Floor	
Target modifier	+
Job impact	Priority
Format	Percentage Usage
Description	Adjusts job priority up when usage falls below target. See the section How Violated Ceilings and Floors Affect Fairshare-Based Priority for more information on how floors affect job priority.

Target type - Target	
Target modifier	N/A

Target type - Target	
Job impact	Priority
Format	Percentage Usage
Description	Adjusts job priority when usage does not meet target.

Setting a fairshare target value of `0` indicates that there is no target and that the priority of jobs associated with that credential should not be affected by the credential's previous fairshare target. If you want a credential's cluster usage near 0%, set the target to a very small value, such as `0.001`.

Example

The following example increases the priority of jobs belonging to user `john` until he reaches 16.5% of total cluster usage. All other users have priority adjusted both up and down to bring them to their target usage of 10%.

```
FSPOLICY          DEDICATEDPS
FSWEIGHT          1
FSUSERWEIGHT      100
USERCFG[john]     FSTARGET=16.5+
USERCFG[DEFAULT]  FSTARGET=10
...
```

5.3.2.B Fairshare Caps

Where fairshare targets affect a job's priority and position in the eligible queue, fairshare caps affect a job's eligibility. Caps can be applied to users, accounts, groups, classes, and QoSes using the `FSCAP` attribute of `*CFG` credential parameters and can be configured to modify scheduling behavior. Unlike fairshare targets, if a credential reaches its fairshare cap, its jobs can no longer run and are therefore removed from the eligible queue and placed in the blocked queue. In this respect, fairshare targets behave like soft limits and fairshare caps behave like hard limits. Fairshare caps can be absolute or relative as described in the following table. If no modifier is specified, the cap is interpreted as relative.

Absolute Cap	
Cap Modifier	^
Job Impact	Feasibility

Absolute Cap	
Format	Absolute Usage
Description	Constrains job eligibility as an absolute quantity measured according to the scheduler charge metric as defined by the FSPOLICY parameter.

Relative Cap	
Cap Modifier	%
Job Impact	Feasibility
Format	Percentage Usage
Description	Constrains job eligibility as a percentage of total delivered cycles measured according to the scheduler charge metric as defined by the FSPOLICY parameter.

Example

The following example constrains the `marketing` account to use no more than `16,500` processor seconds during any given floating one week window. At the same time, all other accounts are constrained to use no more than `10%` of the total delivered processor seconds during any given one week window.

```
FSPOLICY            DEDICATEDPS
FSINTERVAL          12:00:00
FSDEPTH             14
ACCOUNTCFG[marketing]  FSCAP=16500^
ACCOUNTCFG[DEFAULT]    FSCAP=10
...
```

5.3.2.C Priority-Based Fairshare

The most commonly used type of fairshare is priority based fairshare. In this mode, fairshare information does not affect whether a job can run, but rather only the job's priority relative to other jobs. In most cases, this is the desired behavior. Using the standard fairshare target, the priority of jobs of a particular user who has used too many resources over the specified fairshare window is lowered. Also, the standard fairshare target increases the priority of jobs that have not received enough resources.

While the standard fairshare target is the most commonly used, Moab can also specify fairshare ceilings and floors. These targets are like the default target; however, ceilings only adjust priority down when usage is too high and floors only adjust priority up when usage is too low.

Since fairshare usage information must be integrated with Moab's overall priority mechanism, it is critical that the corresponding fairshare priority weights be set. Specifically, the FSWEIGHT component weight parameter and the target type subcomponent weight (such as FSACCOUNTWEIGHT, FSCLASSWEIGHT, FSGROUPWEIGHT, FSQOSWEIGHT, and FSUSERWEIGHT) be specified.

> If these weights are not set, the fairshare mechanism will be enabled but have no effect on scheduling behavior. See 4.1.2 Job Priority Factors for more information on setting priority weights.

Example

```
# set relative component weighting
FSWEIGHT      1
FSUSERWEIGHT  10
FSGROUPWEIGHT 50

FSINTERVAL  12:00:00
FSDEPTH     4
FSDECAY     0.5
FSPOLICY    DEDICATEDPS
# all users should have a FS target of 10%
USERCFG[DEFAULT] FSTARGET=10.0
# user john gets extra cycles
USERCFG[john]    FSTARGET=20.0
# reduce staff priority if group usage exceed 15%
GROUPCFG[staff]  FSTARGET=15.0-
# give group orion additional priority if usage drops below 25.7%
GROUPCFG[orion]  FSTARGET=25.7+
```

> Job preemption status can be adjusted based on whether the job violates a fairshare target using the ENABLEFSVIOLATIONPREEMPTION parameter.

5.3.2.D Credential-Specific Fairshare Weights

Credential-specific fairshare weights can be set using the `FSWEIGHT` attribute of the ACCOUNT, GROUP, and QOS credentials as in the following example:

```
FSWEIGHT  1000
ACCOUNTCFG[orion1] FSWEIGHT=100
ACCOUNTCFG[orion2] FSWEIGHT=200
ACCOUNTCFG[orion3] FSWEIGHT=-100
GROUPCFG[staff] FSWEIGHT=10
```

If specified, a per-credential fairshare weight is added to the global component fairshare weight.

The `FSWEIGHT` attribute is only enabled for ACCOUNT, GROUP, and QOS credentials.

5.3.2.E Fairshare Usage Scaling

Moab uses the `FSSCALINGFACTOR` attribute for QOS credentials to get the calculated fairshare usage of a job:

```
QOSCFG[qos1]   FSSCALINGFACTOR=<double>
```

Moab will multiple the actual fairshare usage by this value to get the calculated fairshare usage of a job. The actual fairshare usage is calculated based on the FSPOLICY parameter.

For an example, if FSPOLICY is set to DEDICATEDPS and a job runs on two processors for 100 seconds, then the actual fairshare usage is 200. If the job ran on a qos with FSSCALINGFACTOR=.5 then Moab multiplies 200*.5=100. If the job ran on a partition with FSSCALINGFACTOR=2 then Moab multiplies 200*2=400.

PARCFG also lets you specify the FSSCALINGFACTOR for partitions. See 6.2.5 Per-Partition Settings.

5.3.2.F Extended Fairshare Examples

Example 5-6: Multi-Cred Cycle Distribution

This example represents a university setting where different schools have access to a cluster. The Engineering department has put the most money into the cluster and therefore has greater access to the cluster. The Math, Computer Science, and Physics departments have also pooled their money into the cluster and have reduced relative access. A support group also has access to the cluster, but since they only require minimal compute time and shouldn't block the higher-paying departments, they are constrained to five percent of the cluster. At this time, users Tom and John have specific high-priority projects that need increased cycles.

```
#global general usage limits - negative priority jobs are considered in scheduling
ENABLENEGJOBPRIORITY   TRUE
# site policy - no job can last longer than 8 hours
USERCFG[DEFAULT] MAX.WCLIMIT=8:00:00
# Note:  default user FS target only specified to apply default user-to-user balance
USERCFG[DEFAULT] FSTARGET=1
# high-level fairshare config
FSPOLICY        DEDICATEDPS
FSINTERVAL      12:00:00
FSDEPTH         32 #recycle FS every 16 days
FSDECAY         0.8 #favor more recent usage info
# qos config
QOSCFG[inst]     FSTARGET=25
QOSCFG[supp]     FSTARGET=5
```

```
QOSCFG[premium] FSTARGET=70
# account config (QoS access and fstargets)
# Note:  user-to-account mapping handled via accounting manager
# Note:  FS targets are percentage of total cluster, not percentage of QOS
ACCOUNTCFG[cs]   QLIST=inst    FSTARGET=10
ACCOUNTCFG[math] QLIST=inst    FSTARGET=15

ACCOUNTCFG[phys] QLIST=supp    FSTARGET=5
ACCOUNTCFG[eng]  QLIST=premium FSTARGET=70
# handle per-user priority exceptions
USERCFG[tom]  PRIORITY=100
USERCFG[john] PRIORITY=35
# define overall job priority
USERWEIGHT       10  # user exceptions
# relative FS weights (Note: QOS overrides ACCOUNT which overrides USER)
FSUSERWEIGHT     1
FSACCOUNTWEIGHT  10
FSQOSWEIGHT      100
# apply XFactor to balance cycle delivery by job size fairly
# Note:  queuetime factor also on by default (use QUEUETIMEWEIGHT to adjust)
XFACTORWEIGHT    100
# enable preemption
PREEMPTPOLICY   REQUEUE
# temporarily allow phys to preempt math
ACCOUNTCFG[phys] JOBFLAGS=PREEMPTOR PRIORITY=1000
ACCOUNTCFG[math] JOBFLAGS=PREEMPTEE
```

5.3.3 Hierarchical Fairshare/Share Trees

Moab supports arbitrary depth hierarchical fairshare based on a share tree. In this model, users, groups, classes, and accounts can be arbitrarily organized and their usage tracked and limited. Moab extends common share tree concepts to allow mixing of credential types, enforcement of ceiling and floor style usage targets, and mixing of hierarchical fairshare state with other priority components.

You can terminate your tnode with '</tnode>' or '<tnode />'.

In this topic:

5.3.3.A Defining the Tree

The FSTREE parameter can be used to define and configure the share tree used in fairshare configuration. This parameter supports the following attributes:

SHARES	
Format	`<COUNT>[@<PARTITION>][,<COUNT>[@<PARTITION>]]...`, where `<COUNT>` is a double and `<PARTITION>` is a specified partition name.
Description	The node target usage or share.

Example	`FSTREE[Eng]   SHARES=1500.5`
	`FSTREE[Sales] SHARES=2800`

MEMBERLIST	
Format	Comma-delimited list of child nodes of the format `[<OBJECT_TYPE>]:<OBJECT_ID>` where object types are only specified for *leaf nodes* associated with **user**, **group**, **class**, **qos**, or **acct** credentials.
Description	The tree objects associated with this node.

Example	`FSTREE[root]   SHARES=100      MEMBERLIST=Eng,Sales`
	`FSTREE[Eng]    SHARES=1500.5   MEMBERLIST=user:john,user:steve,user:bob`
	`FSTREE[Sales]  SHARES=2800     MEMBERLIST=Sales1,Sales2,Sales3`
	`FSTREE[Sales1] SHARES=30       MEMBERLIST=user:kellyp,user:sam`
	`FSTREE[Sales2] SHARES=10       MEMBERLIST=user:ux43,user:ux44,user:ux45`
	`FSTREE[Sales3] SHARES=60       MEMBERLIST=user:robert,user:tjackson`

Current tree configuration and monitored usage distribution is available using the mdiag -f -v commands.

5.3.3.B Controlling Tree Evaluation

Moab provides multiple policies to customize how the share tree is evaluated.

Policy	Description
FSTREETIERMULTIPLIER	Decreases the value of sub-level usage discrepancies. It can be a positive or negative value. When positive, the parent's usage in the tree takes precedence; when negative, the child's usage takes precedence. The usage amount is not changed, only the coefficient used when calculating the value of fstree usage in priority. When using this parameter, we recommend that you research how it changes the values in `mdiag -p` to determine the appropriate use.
FSTREECAP	Caps lower level usage factors to prevent them from exceeding upper tier discrepancies.

Using FS Floors and Ceilings with Hierarchical Fairshare

All standard fairshare facilities including target floors, target ceilings, and target caps are supported when using hierarchical fairshare.

Multi-Partition Fairshare

Moab supports independent, per-partition hierarchical fairshare targets allowing each partition to possess independent prioritization and usage constraint settings. This is accomplished by setting the `PERPARTITIONSCHEDULING` attribute of the `FSTREE` parameter to `TRUE` in `moab.cfg` and setting `partition="name"` in your `<fstree>` leaf.

```
FSTREE[tree]
  <fstree>
    <tnode partition="slave1" name="root" type="acct" share="100" limits="MAXJOB=6">
      <tnode name="accta" type="acct" share="50" limits="MAXSUBMITJOBS=2 MAXJOB=1">
        <tnode name="fred" type="user" share="1" limits="MAXWC=1:00:00">
        </tnode>
      </tnode>
      <tnode name="acctb" type="acct" share="50" limits="MAXSUBMITJOBS=4 MAXJOB=3">
        <tnode name="george" type="user" share="1" >
        </tnode>
      </tnode>
    </tnode>
    <tnode partition="slave2" name="root" type="acct" share="100"
limits="MAXSUBMITJOBS=6 MAXJOB=5">
      <tnode name="accta" type="acct" share="50">
        <tnode name="paul" type="user" share="1">
        </tnode>
      </tnode>
      <tnode name="acctb" type="acct" share="50">
        <tnode name="ringo" type="user" share="1">
        </tnode>
      </tnode>
    </tnode>
  </fstree>
```

If no partition is specified for a given share value, then this value is assigned to the global partition. If a partition exists for which there are no explicitly specified shares for any node, this partition will use the share distribution assigned to the global partition.

Dynamically Importing Share Tree Data

Share trees can be centrally defined within a database, flat file, information service, or other system and this information can be dynamically imported and used within Moab by setting the `FSTREE` parameter within the Identity Managers. This interface can be used to load current information at startup and periodically synchronize this information with the master source.

To Create a Fairshare Tree in a Separate XML File and Import it into Moab

1. Create a file to store your fair share tree specification. Give it a descriptive name and store it in your Moab home directory (`$MOABHOMEDIR` or `$MOABHOMEDIR/etc`). In this example, the file is called `fstree.dat`.
2. In the first line of `fstree.dat`, set `FSTREE[myTree]` to indicate that this is a fairshare file.
3. Build a tree in XML to match your needs. For example:

```
FSTREE[myTree]
<fstree>
<tnode name="root" share="100">
<tnode name="john" type="user" share="50" limits="MAXJOB=8 MAXPROC=24
MAXWC=01:00:00"></tnode>
<tnode name="jane" type="user" share="50" limits="MAXJOB=5"></tnode>
</tnode>
</fstree>
```

   This configuration creates a fairshare tree where users share a value of 100. Users `john` and `jane` share the value equally, because each has been given 50.

   Because 100 is an arbitrary number, users `john` and `jane` could be assigned `10000` and `10000` respectively and still have a 50% share under the parent leaf. To keep the example simple, however, we recommend that you use 100 as your arbitrary share value and distribute the share as percentages. In this case, `john` and `jane` each have 50%.

   If the users' numbers do not add up to at least the fairshare value of 100, the remaining value is shared among all users under the tree. For instance, if the tree had a value of 100, user `john` had a value of 50, and user `jane` had a value of 25, then 25% of the fairshare tree value belongs to all other users associated with the tree. By default, tree leaves do not limit who can run under them.

   Each value specified in the `tnode` elements must be contained in quotation marks.

4. (Optional) Share trees defined within a flat file can be cumbersome; consider running tidy for xml to improve readability. Sample usage:

```
> tidy -i -xml mam-tiy.cfg <filename> <output file>

# Sample output

FSTREE[myTree]
<fstree>
  <tnode name="root" share="100">
    <tnode name="john"   type="user" share="50" limits="MAXJOB=8
    MAXPROC=24 MAXWC=01:00:00">
    </tnode>
    <tnode name="jane" type="user" share="50" limits="MAXJOB=5">
    </tnode>
  </tnode>
```

```
</fstree>
```

5. Link the new file to Moab using the IDCFG parameter in your Moab configuration file:

   ```
   IDCFG[myTree] server="FILE:///$MOABHOMEDIR/etc/fstree.dat" REFRESHPERIOD=INFINITY
   ```

   Moab imports the `myTree` fairshare tree from the `fstree.dat` file. Setting `REFRESHPERIOD` to `INFINITY` causes Moab to read the file each time it starts or restarts, but setting a positive interval (e.g., 4:00:00) cause Moab to read the file more often. See 18.4.6 Refreshing Identity Manager Data for more information.

6. To view your fairshare tree configuration, run mdiag -f. If it is configured correctly, the tree information will appear beneath all the information about your fairshare settings configured in `moab.cfg`.

   ```
   > mdiag -f
   Share Tree Overview for partition 'ALL'
   Name          Usage    Target                (FSFACTOR)
   ----          ------   ------               -------------
   root        100.00    100.00  of   100.00 (node: 1171.81) (0.00)
   - john       16.44     50.00  of   100.00 (user: 192.65) (302.04) MAXJOB=8
   MAXPROC=24 MAXWC=3600
   - jane       83.56     50.00  of   100.00 (user: 979.16) (-302.04)  MAXJOB=5
   ```

   The settings you configured in fstree.dat appear in the output. The tree of 100 is shared equally between users `john` and `jane`.

Specifying Share Tree Based Limits

Limits can be specified on internal nodes of the share tree using standard credential limit semantics. The following credential usage limits are valid:

- MAXIJOB (maximum number of idle jobs allowed for the credential)
- MAXJOB
- MAXMEM
- MAXNODE
- MAXPROC
- MAXSUBMITJOBS
- MAXWC

Example 5-7: FSTREE limits example

```
FSTREE[myTree]
<fstree>
  <tnode name="root" share="100">
   <tnode name="john"   type="user" share="50" limits="MAXJOB=8
    MAXPROC=24 MAXWC=01:00:00">
    </tnode>
    <tnode name="jane" type="user" share="50" limits="MAXJOB=5">
    </tnode>
```

```
  </tnode>
</fstree>
```

Specifying a Default Account in a Fair Share Tree

The adef attribute in a fair share tree can be used to specify a default account for a credential and its children. This is useful for sites with many users who need access to an account and who use an identity manager to import credentials.

The rules are as follows:

- When you define an adef attribute on a credential such as a user or qos, then the child under the tnode inherits the credential.
- When a child has an adef, then that adef overrides the parent.

To define an adef, add a qdef attribute to the tnode for the user:

```
<tnode name="jane" type="user" adef="acct2" share="1"
qlist="batch,special,standby,test,exempt,expedite,super"> </tnode>
```

Other Uses of Share Trees

If a share tree is defined, it can be used for purposes beyond fairshare, including organizing general usage and performance statistics for reporting purposes (see 3.7.36 showstats), enforcement of tree node based usage limits, and specification of resource access policies.

Related Topics

- 3.7.8 mdiag -f (command that provides diagnosis and monitoring of the fairshare facility)
- FSENABLECAPPRIORITY parameter
- ENABLEFSPREEMPTION parameter
- FSTARGETISABSOLUTE parameter

5.4 Sample FairShare Data File

FS.<EPOCHTIME>

```
# FS Data File (Duration:  43200 seconds)  Starting: Sat Jul  8 06:00:20
user                jvella   134087.910
user              reynolds    98283.840
user                gastor    18751.770
user                uannan   145551.260
user              mwillis   149279.140
...
group             DEFAULT   411628.980
group             RedRock  3121560.280
group              Summit   500327.640
group              Arches  3047918.940
acct        Administration   653559.290
acct           Engineering  4746858.620
acct                Shared    75033.020
acct              Research  1605984.910
qos               Deadline  2727971.100
qos           HighPriority  4278431.720
qos               STANDARD    75033.020
class                batch  7081435.840
sched             iCluster  7081435.840
```

The total usage consumed in this time interval is 7081435.840 processor-seconds. Since every job in this example scenario had a user, group, account, and QOS assigned to it, the sum of the usage of all members of each category should equal the total usage value: USERA + USERB + USERC + USERD = GROUPA + GROUPB = ACCTA + ACCTB + ACCTC = QOS0 + QOS1 + QOS2 = SCHED.

5.5 Accounting, Charging, and Allocation Management

In this section:

For a complete list of and additional information on the AMCFG parameters and flags, see 5.6 AMCFG Parameters and Flags.

5.5.1 Accounting Manager

An accounting manager is a software system that enables tracking and charging for job resource usage. Moab Accounting Manager is a commercial charge-back accounting system that has built-in integration with Moab Workload Manager. Moab Accounting Manager can be used in a variety of accounting modes such as for usage tracking, notional charging or allocation enforcement.

When used for usage tracking only, the accounting manager simply records workload usage details. When configured additionally to perform charging, resource charge rates are used to impute a charge for each job. When configured to enforce resource allocation limits, jobs are charged against allocations and new jobs may be blocked from running if their account runs out of funds. See the section Accounting Mode below and also see 'Select an Appropriate Accounting Mode' in the *Moab Accounting Manager Administrator Guide* for more information on supported accounting modes.

In a typical allocation enforcement use case, credits are allocated to accounts for designated time periods; establishing limits on the use of compute resources. The base currency credits can be defined in terms of system resource units (e.g., Processor-Seconds) or a real currency (e.g., U.S. dollars). Charge rates are established for the use of resources. Accounts are created and users are given access to the appropriate accounts. Deposits are made into funds associated with the account’s creating allocations. An allocation cycle can be established whereby funds are reset on a regular periodic basis (such as yearly, quarterly, or monthly) and where allocations are renewed for accepted accounts. Before a job is started, Moab Workload Manager will verify that the user has sufficient credits to run the job by attempting to place a hold against their funds (referred to as a lien). When a job completes, the user's funds will be debited via a charge, usage information will be recorded for the job, and the lien will be removed.

5.5.2 Accounting Mode

The accounting mode (specified via the AMCFG[] MODE parameter) modifies the way in which accounting-relevant job and reservation stages (e.g., create, start, end, etc.) are processed. See 5.5.7 Accounting Stages for more information on the behaviors of the different values of the accounting mode.

The following table describes the valid values for the accounting mode:

Value	Description
strict-allocation	Use this mode if you want to strictly enforce allocation limits. Under this mode, holds (called liens) will be placed against allocations in order to prevent multiple jobs from starting up on the same funds. Jobs and reservations can be prevented from running if the end-users do not have

Value	Description
	sufficient funds. This is the default.
fast-allocation	Use this mode if you want to debit allocations, but need higher throughput by eliminating the lien and quote operations of strict-allocation mode. Under this mode, jobs and reservations check a cached account balance, and can be prevented from running after the balance has become zero or negative. The cached account balances in Moab can be viewed by running 'mrmctl -q AccountBalanceCache AM'. If you are using fast-allocation, funds are assumed to have account-based constraints only. Moab will reject funds having no constraints or having non-account constraints. It is highly recommended that you enable `ENFORCEACCOUNTACCESS` to `TRUE` and *`AMCFG[] CREATECRED=TRUE`* with an appropriate refresh period (via *`AMCFG[] REFRESHPERIOD`*) so that Moab can prevent jobs from running under accounts that the user does not belong to (this is enforced via liens in the strict-allocation accounting mode). Also, the configured refresh period will apply to both credential updates and account balance updates. See Appendix A: Moab Parameters for more information on the `ENFORCEACCOUNTACCESS` parameter.
notional-charging	Use this mode if you want to calculate and record charges for workload usage, but not keep track of fund balances or allocation limits.
usage-tracking	Use this mode if you want to record workload usage details, but not to calculate a charge nor keep track of fund balances or allocation limits.

5.5.3 Accounting Manager Interface Types

Moab Workload Manager supports two accounting manager interface types:

- MAM - When using the MAM Accounting Manager interface type, Moab communicates directly over the network with Moab Accounting Manager using the SSS wire protocol.
- Native - When using the Native Accounting Manager interface type, Moab invokes scripts that can be customized to interact with Moab Accounting Manager or other third party accounting systems.

5.5.3.A MAM

The MAM Accounting Manager interface type enables direct communication between Moab Workload Manager and Moab Accounting Manager. This often results in the fastest accounting performance. Use this interface type if you do not need to customize the interaction with the accounting manager.

To configure Moab to use the MAM Accounting Manager interface, run `configure` using the `--with-am` option.

Example 5-8: configure --with-am

```
./configure --with-am=mam ...
```

Consequently, make `install` will add the essential configuration and connection entries into the `moab.cfg` and `moab-private.cfg` files.

The following are typical entries in the Moab configuration files for using the MAM interface:

- `moab.cfg`

  ```
  AMCFG[mam] TYPE=MAM HOST=localhost
  ```

- `moab-private.cfg`

  ```
  CLIENTCFG[AM:mam] KEY=UiW7EihzKyUyVQg6dKirDhV3
  ```

Synchronize the secret key with Moab Accounting Manager by copying the value of the `token.value` parameter from the `MAM_PREFIX/etc/mam-site.conf` file, which is randomly generated during the Moab Accounting Manager install process.

When using the MAM Accounting Manager interface, by default Moab will communicate directly with Moab Accounting Manager via the SSS wire protocol. However, it is possible to enable a hybrid model and override individual accounting actions by specifying the exec protocol and the path of a custom script to the appropriate AMCFG[] *URL parameters.

Moab Accounting Manager should be installed, started, and initialized. See 'Initial Setup' in the *Moab Accounting Manager Administrator Guide* for examples of how to initialize MAM for your initial mode of operation.

5.5.3.B Native

The Native Accounting Manager interface type provides a customization layer between Moab Workload Manager and Moab Accounting Manager. This interface can be used where greater accounting customization is required. This interface can also be customized to interact with third-party accounting manager systems. Moab passes job accounting details to scripts that handle the interaction with the external system.

To configure Moab to use the MAM Accounting Manager interface, run configure using the `--with-am=native` option.

Additionally, you might need to use the `--with-am-dir` configure option to specify the prefix directory for Moab Accounting Manager if MAM has been installed in a non-default location.

Example 5-9: configure --with-am=native

```
./configure --with-am=native ...
```

Consequently, make `install` will add the essential accounting manager entries into `moab.cfg` and install the accounting-related scripts (`$PREFIX/tools/mam/usage.*.mam.pl`) in the correct locations.

Moab will default to using a set of stock scripts for the accounting stages. To view the scripts that are currently in use, run mdiag -R -v am: (even more information may be available in `mdiag -R -v --xml`). The following shows sample output from running the *`mdiag -R -v am:`* command:

```
AM[mam]  Type:  Native  State: 'Active'
 Timeout:                            15
 Accounting Mode:                    strict-allocation
 Job Charge Policy:                  All
 Reservation Charge Policy:          Select
 Create URL:                         exec:///opt/moab/tools/mam/usage.quote.mam.pl
 Start URL:                          exec:///opt/moab/tools/mam/usage.reserve.mam.pl
 Pause URL:                          exec:///opt/moab/tools/mam/usage.charge.mam.pl
 Resume URL:                         exec:///opt/moab/tools/mam/usage.reserve.mam.pl
 Update URL:                         exec:///opt/moab/tools/mam/usage.charge.mam.pl
 Continue URL:                       exec:///opt/moab/tools/mam/usage.reserve.mam.pl
 End URL:                            exec:///opt/moab/tools/mam/usage.charge.mam.pl
 Delete URL:                         exec:///opt/moab/tools/mam/lien.delete.mam.pl
 Query URL:                          exec:///opt/moab/tools/mam/account.query.mam.pl
 Retry Failed Charges:               TRUE
```

Moab will invoke the Native Accounting Manager scripts by passing the job or reservation information via XML to the standard input of the script. You can override any of the default scripts with a custom script by specifying the appropriate AMCFG URL parameter in the moab server configuration file. See 5.6 AMCFG Parameters and Flags for `CREATEURL`, `STARTURL`, `PAUSEURL`, `RESUMEURL`, `UPDATEURL`, `CONTINUEURL`, `ENDURL`, `DELETEURL`, and `QUERYURL` values for more information.

The XML sent to the scripts is in the form of an SSS Request that is identical to the Request sent to MAM when you use the MAM Accounting Manager interface type. For example, the XML sent to the usage.charge.mam.pl script in a final charge consists of an encapsulating Request element with an action attribute that has a value of 'Charge'; an object element with a value of 'UsageRecord'; one or more optional Option elements; and a Data element. The Data element has a single UsageRecord element with property elements describing the job or reservation properties. For example:

```
<Request action="Charge"><Object>UsageRecord</Object><Option
```

```
name="Duration">1234</Option><Data><UsageRecord><Type>Job</Type><Instance>Moab.165</In
stance><User>amy</User><Group>staff</Group><Account>chemistry</Account><Class>batch</C
lass><QualityOfService>high</QualityOfService><Machine>colony</Machine><Nodes>1</Nodes
><NodeType>Fast</NodeType><NodeCharge>2.000000</NodeCharge><Partition>Torque</Partitio
n><Processors
consumptionRate="0.50">2</Processors><Memory>2048</Memory><Matlab>2</Matlab><StartTime
>1398805354</StartTime><EndTime>1398805357</EndTime><ExitCode>0</ExitCode><OpSys>CentO
S</Opsys><Temp>87.00</Temp></UsageRecord></Data></Request> *
```

In the sample XML above, Matlab is an example of a generic resource, Opsys is an example of a job variable, and Temp is an example of a generic metric.

A reservation charge, or quote or lien, is very similar. For example:

```
<Request action="Charge"><Object>UsageRecord</Object><Option
name="Duration">7200</Option><Data><UsageRecord><Type>Reservation</Type><Instance>rese
rvation.7</Instance><User>amy</User><Machine>colony</Machine><Nodes>1</Nodes><Processo
rsconsumptionRate="0.76">12</Processors><Duration>7200</Duration><StartTime>1398797430
</StartTime><EndTime>1398804630</EndTime></UsageRecord></Data></Request>
```

The majority of the scripts use this same basic XML format; for instance, `usage.quote.mam.pl`, `usage.reserve.mam.pl`, and `usage.charge.mam.pl`.

The XML sent to the lien.delete.mam.pl script to clean up after a failure consists of an encapsulating Request element with an action attribute that has a value of 'Delete'; an object element with the value of 'Lien'; and a condition (Where) element indicating the lien instance to delete. For example:

```
<Request action="Delete"><Object>Lien</Object><Where
name="Instance">Moab.127</Where></Request>
```

The script should return a return code (zero for success), data on standard out and messages on standard error. A failure in `CREATEURL`, `STARTURL`, `RESUMEURL`, or `CONTINUEURL` should result in the application of the `CREATEFAILUREACTION`, `STARTFAILUREACTION`, `RESUMEFAILUREACTION`, or `CONTINUEFAILUREACTION` respectively.

Moab Accounting Manager should be installed, started, and initialized. The simplest procedure is to install it on the same server as Moab Workload Manager so that the Moab Accounting Manager can share libraries and configuration files with Moab Workload Manager and Moab Accounting Manager scripts. See 'Initial Setup' in the *Moab Accounting Manager Administrator Guide* for examples of how to initialize MAM for your initial mode of operation.

5.5.4 Charging for Jobs

Moab can be configured to charge for the resources used in jobs.

Job tracking and charging via an accounting manager is enabled or disabled by the `AMCFG []` JOBCHARGEPOLICY parameter. By default, if an accounting manager is defined in Moab, Moab will charge for all jobs, independent of the job’s exit status (derived from an internal default of AMCFG[] JOBCHARGEPOLICY=All). This is the policy that is most frequently used because of the unfortunate fact that a clever user can fake a job failure to avoid being charged for their job. Jobs that are discovered to have failed due to system issues can be proactively refunded by the system admin, or the user may be required to apply for a refund. Charges can be restricted to jobs that complete successfully by setting AMCFG[] JOBCHARGEPOLICY=Successful. To disable charging for jobs entirely, set AMCFG[] JOBCHARGEPOLICY=None.

When using the default accounting mode of strict-allocation, before Moab starts a job, it contacts the accounting manager and requests an allocation reservation (or lien) be placed on the associated account. The lien amount is equivalent to the total cost of resources that could be consumed by the job (based on the job's wallclock limit) and is used to prevent the possibility of allocation oversubscription. Moab then starts the job. When the job completes, Moab debits the allocation by the amount actually consumed by the job and then releases the lien.

These steps should be transparent to users. Only when an account has insufficient allocations to run a requested job will the presence of the accounting manager be noticed. The policies guiding what action should be taken if a user is out of funds is dictated by the AMCFG[] STARTFAILUREACTION (and related) parameters. If desired, a fallback account can be specified for use when a job's primary account is out of allocations. This account, specified using the `AMCFG` parameter's FALLBACKACCOUNT attribute, is often associated with a low QoS privilege, priority, and cost and is often configured to run only when no other jobs are present.

The actual policies that determine what resources are charged for and in what amounts are specified by the charge rates defined in the accounting manager. Moab will pass the job properties (shown in 5.5.6 Accounting Properties Reported to the Accounting Manager) to the accounting manager. It is the task of the accounting manager to record and charge for the job according to site objectives.

5.5.5 Charging for Reservations

Moab can be configured to charge for the unused cycles in reservations. One of the hesitations with dedicating resources to a particular group is that if the resources are not used by that group, they go idle and are wasted. By configuring a reservation to be chargeable, sites can charge every idle cycle of the reservation to a particular account. When the reservation is in use, the consumed resources will be charged to the jobs using the resources. When the resources are idle, the resources will be charged to the reservation's charge account. See `AMCFG[]` RESERVATIONCHARGEPOLICY.

If `RESERVATIONCHARGEPOLICY` is set to `Select` (the default), charging can be enabled for select reservations by specifying the CHARGEACCOUNT and CHARGEUSER attributes for the reservation. For standing reservations, these are set via the SRCFG[X]

CHARGEACCOUNT and CHARGEUSER parameters. For administrative reservations, these are set via the -S aaccount and auser options.

Example 5-10: Enabling charging in a standing reservation

```
SRCFG[foo] PERIOD=DAY DAYS=Mon,Tue,Wed,Thu,Fri DEPTH=1 USERLIST=amy
CHARGEACCOUNT=chemistry CHARGEUSER=amy RESOURCES=PROCS:1 TASKCOUNT=2
```

Example 5-11: Enabling charging in an administrative reservation

```
mrsvctl -c -a USER=amy -S aaccount=chemistry -S auser=amy -R procs=1 -t 1 -d 7200
```

If RESERVATIONCHARGEPOLICY is set to All, idle cycles will be charged for all reservations unless disabled for individual reservations by specifying the reservation Charge attribute with a value of False. For standing reservations, this are set via the SRCFG CHARGE parameter. For administrative reservations, this is set via the -S charge options.

Example 5-12: Disabling charging in a standing reservation

```
SRCFG[foo] PERIOD=DAY DAYS=Mon,Tue,Wed,Thu,Fri DEPTH=1 USERLIST=amy CHARGE=False
RESOURCES=PROCS:1 TASKCOUNT=2
```

Example 5-13: Disabling charging in an administrative reservation

```
mrsvctl -c -a USER=amy -S charge=False -R procs=1 -t 1 -d 7200
```

If RESERVATIONCHARGEPOLICY is set to None, idle cycles will not be charged for any reservations, regardless of what attributes are specified with the reservation.

When utilizing the accounting manager to track or charge for idle cycles in any reservations, the accounting manager must be configured to track and charge for reservation-relevant properties. See the examples for 'Enabling Reservation Statistics' and 'Charging for the unused cycles in reservations' in the *Moab Accounting Manager Administrator Guide* for steps on how to do this for the Moab Accounting Manager.

5.5.6 Accounting Properties Reported to the Accounting Manager

Moab can send the following information to the accounting manager via charging actions.

In this topic:

5.5.6.A For Jobs

Property name in the Accounting Manager Usage Record	Description of property value recorded in the Accounting Manager Usage Record
Account	Account name.
BlockedProcessors*	Number of processors blocked from use to other jobs. This might be more than the allocated or requested processors if entire nodes are given exclusive access to the job (e.g., from a node access policy or node exclusivity flag). Tracking or charging for BlockedProcessors should only be used with policies that allow only a single job to dedicate a node such as with a Node Access Policy of SINGLEJOB or SINGLETASK, or using a QOS with the DEDICATED flag. Using BlockedProcessors with any policy allowing more than one job to dedicate a node (such as a Node Access Policy of SINGLEUSER, SINGLECLASS, SINGLEACCOUNT or UNIQUEUSER) will yield inconsistent results and is not recommended or supported.
Charge	If the `AMCFG` LOCALCOST flag is set, Moab will calculate and pass the Charge amount to MAM. If it is not, MAM will calculate the charge based on the transmitted job properties.
Class	Class/queue name.
Cores*	NUMA cores allocated to the job. Cores will be reported to the accounting manager when using the -L NUMA Resource Request syntax and specifying place=socket, place=numanode or place=core.
CPUTime	CPU time. The value sent by Moab to the accounting manager is the cumulative CPU time for the job. Using this value as the basis of charging is not compatible with periodic charging.
Duration	Moab sends the wallclock time for the job charge(s) in seconds. This is aggregated in MAM as Duration.
EndTime	Job end time.

Property name in the Accounting Manager Usage Record	Description of property value recorded in the Accounting Manager Usage Record
ExitCode*	Exit code.
Features*	Allocated node features. Node features are sent in the JSON object format `{"<node_feature_or_combo_name>":<node_feature_or_combo_count>,...}`. Node features are not passed to the accounting manager unless configured to do so via the AMCFG[] IncludeFeatures parameter.
GPUs*	Number of GPUs allocated to the job.
Group	Group name.
Instance	Job ID.
Licenses	License generic resources allocated to the job. License generic resources are sent in the JSON object format `{"<license_generic_resource_name>":<license_generic_resource_value>,...}`. The license generic resource value is the number of this license generic resource consumed by the job. License generic resources are distinguished by those generic resources having the license trait. Generic resources can be marked as license generic resources in two primary ways: • Any generic resources seen on nodes reported via a Native Resource Manager having the server configuration parameter RMCFG[] RESOURCETYPE=License will be marked as a license generic resource. • Generic resources defined in Moab can be identified as license types by using the Moab configuration parameter GRESCFG[] LICENSE=TRUE. You can distinguish the generic resources having the license trait via the command `mschedctl -l gres -v`.
Machine	Cluster (resource manager) name.
Memory	Dedicated or utilized memory in megabytes.
Metrics	Generic metrics are sent in the JSON object format `{"<generic_`

Property name in the Accounting Manager Usage Record	Description of property value recorded in the Accounting Manager Usage Record
	`metric_name>":<generic_metric_value>,...}` where the generic metric value is the average value of the generic metric across the nodes of the job and across time.
MICs*	Number of MICs allocated to the job.
NodeCharge*	Aggregate node charge rate. See the parameters NODECHARGEPOLICY and CHARGERATE for more information.
Nodes	Node count.
NodeType*	Node type. See the node attribute NODETYPE for more information.
NumaNodes*	NUMA nodes allocated to the job. NumaNodes will be reported to the accounting manager when using the -L NUMA Resource Request syntax and specifying place=socket or place=numanode.
Partition*	Partition name.
ProcessorEquivalents*	Processor Equivalents.
Processors	Number of processors allocated to the job. This is normally equivalent to the requested processors.
QualityOfService	QoS name.
QueueDuration*	Effective duration the job was in the idle state.
RequestedDuration	Requested wallclock limit.
Resources	Generic resources are sent in the JSON object format `{"<generic_resource_name>":<generic_resource_value>,...}`. The generic resource value is the number of this generic resource consumed by the job.
Sockets*	NUMA sockets allocated to the job. Sockets will be reported to the accounting manager when using the -L NUMA Resource Request syntax and specifying place=socket.

Property name in the Accounting Manager Usage Record	Description of property value recorded in the Accounting Manager Usage Record
Stage	Accounting stage.
StartTime	Job start time.
SubmitTime	Job submission time.
Threads*	NUMA threads allocated to the job. Threads will be reported to the accounting manager when using the -L NUMA Resource Request syntax and specifying place=socket, place=numanode, place=core or place=thread.
Type	Set to 'Job'.
User	User name.
Variables	Job variables are sent in the JSON object format `{"<job_variable_name>":"<job_variable_value>",...}`.

* For this property to be recorded in the MAM Usage Record, you must define a custom usage record attribute in MAM for it. See 'Customizing the Usage Record Object' in the *Moab Accounting Manager Administrator Guide* for more information.

5.5.6.B For Reservations

Property name in MAM Usage Record	Description of property value recorded in MAM Usage Record
Account	Charge account.
Duration	Moab sends the wallclock time for the reservation in seconds. This is aggregated in MAM as Duration.
EndTime	Reservation end time.
IdleProcessorSeconds	Processor seconds not blocked by jobs within the reservation. This is the metric typically used to charge for spare cycles not attributed to jobs within the reservation.

Property name in MAM Usage Record	Description of property value recorded in MAM Usage Record
Instance	Reservation ID.
Machine	Cluster (resource manager) name.
Nodes	Number of node allocated to the reservation.
Partition	Partition name.
Processors	Number of processors allocated to the reservation.
ReservedProcessorSeconds	Total processor seconds included within the reservation (to-date).
Stage	Accounting stage.
StartTime	Reservation start time.
Type	Set to 'Reservation'.
User	Charge user or reservation owner.

5.5.7 Accounting Stages

The accounting manager performs various actions throughout different stages of a job or reservation lifetime. For a stock configuration (meaning you have not overridden the accounting actions with custom scripts), the following describes the stages and the respective actions that occur at these stages depending on the accounting mode:

- **Create stage** – When a job is submitted or a chargeable reservation is created and either AMCFG[] VALIDATEJOBSUBMISSION is TRUE or an AMCFG[] FALLBACKACCOUNT or FALLBACKQOS are specified:
    - If the accounting mode is `strict-allocation`, Moab will check with the accounting manager to verify that sufficient funds exist for the job or reservation to run.
    - If the accounting mode is `fast-allocation`, Moab will check its cached balance for the job's or reservation's account, to verify that sufficient funds exist for the fund or reservation to run.
    - Otherwise, it does nothing.

- **Start stage** – When a job or a chargeable reservation is about to start:
    - If the accounting mode is `strict-allocation`, Moab will attempt to place a hold against the allocation in the accounting manager in order to prevent multiple jobs or reservations from starting on the same funds.
    - If the accounting mode is `fast-allocation`, Moab will check its cached balance for the job's or reservation's account, to verify that sufficient funds exist for the job or reservation to run.*
    - Otherwise, it does nothing.
- **Delete stage** – If a job or chargeable reservation fails to start:
    - If the accounting mode is `strict-allocation` and Moab has already placed a hold on an allocation for the job or reservation, Moab will contact the accounting manager to remove the lien.
    - Otherwise, it does nothing.
- **Pause stage** – If a job becomes suspended, Moab will make a charge for the resources used for the time the job has run this far:
    - If the accounting mode is `strict-allocation`, the usage record will be updated with resource usage and charge amounts, the allocation will be debited, and the lien will be reduced.
    - If the accounting mode is `fast-allocation`, the usage record will be updated with resource usage and charge amounts, and the allocation will be debited.
    - If the accounting mode is `notional-charging`, the usage record will be updated with resource usage and charge amounts.
    - If the accounting mode is `usage-tracking`, the usage record will be updated with resource usage.
- **Resume stage** – If a suspended job is resumed:
    - If the accounting mode is `strict-allocation`, Moab will attempt to place a hold against the funds in the accounting manager for the smaller of (the duration of the next charge period, or the remaining duration of the job or reservation).
    - If the accounting mode is `fast-allocation`, Moab will check its cached balance for the job's or reservation's account, to verify that sufficient funds exist for the job or reservation to run for the smaller of (the duration of the next charge period, or the remaining duration of the job or reservation).*
    - Otherwise, it does nothing.
- **Update stage** – If AMCFG[] FLUSHINTERVAL is set and Moab has reached the end of a charge period, Moab will make an incremental charge for all running jobs and

active chargeable reservations for the resources used during the last charge period:

  - If the accounting mode is `strict-allocation`, the usage record will be updated with resource usage and charge amounts, the allocation will be debited, and the lien will be reduced.
  - If the accounting mode is `fast-allocation`, the usage record will be updated with resource usage and charge amounts, and the allocation will be debited.
  - If the accounting mode is `notional-charging`, the usage record will be updated with resource usage and charge amounts.
  - If the accounting mode is `usage-tracking`, the usage record will be updated with resource usage.

- **Continue stage** – If AMCFG[] FLUSHINTERVAL is set and Moab is beginning a new charge period for a job or reservation:
  - If the accounting mode is `strict-allocation`, Moab will attempt to place a hold against the funds in the accounting manager for the smaller of (the duration of the next charge period, or the remaining duration of the job or reservation).
  - If the accounting mode is `fast-allocation`, Moab will check its cached balance for the job's or reservation's account, to verify that sufficient funds exist for the job or reservation to run for the smaller of (the duration of the next charge period, or the remaining duration of the job or reservation.*
  - Otherwise, it does nothing.
- **End stage** – If a job or chargeable reservation ends, Moab will make a final charge for the remainder of the resources used by the job or reservation:
  - If the accounting mode is `strict-allocation`, the usage record will be updated with resource usage and charge amounts, the allocation will be debited, and the lien will be removed.
  - If the accounting mode is `fast-allocation`, the usage record will be updated with resource usage and charge amounts, and the allocation will be debited.
  - If the accounting mode is `notional-charging`, the usage record will be updated with resource usage and charge amounts.
  - If the accounting mode is `usage-tracking`, the usage record will be updated with resource usage.

* The cached account balances in Moab can be viewed by running 'mrmctl -q AccountBalanceCache AM'.

5.5.8 Accounting Events

You can add accounting events to the event log by specifying one or more of the following with `RECORDEVENTLIST`:

Event	Description
AMCREATE	Record accounting events triggered when an object is created; for example, when a balance check occurs at job submission.
AMDELETE	Record accounting events triggered when an object's normal accounting lifecycle is interrupted; for example, when the lifecycle is interrupted to clean up reservations for a failed job start.
AMEND	Record accounting events triggered when an object ends; for example, when a charge occurs at the end of a job.
AMPAUSE	Record accounting events triggered when an object is paused; for example, when a partial charge occurs when a job is paused.
AMQUOTE	Record accounting events triggered when an object requires a quote amount.
AMRESUME	Record accounting events triggered when an object is resumed; for example, when a lien is made when a job is resumed.
AMSTART	Record accounting events triggered when an object is started; for example, when a lien is made when a job starts.
AMUPDATE	Record accounting events triggered when an object continues past a flush interval; for example, when a partial charge occurs and new lien is made for a job.

5.5.9 Blocking Versus Non-Blocking Accounting Actions

Moab uses a thread pool to perform non-blocking actions. Instead of blocking the scheduling thread, the request is added to a queue that is serviced by the accounting thread pool. Using the thread pool to perform non-blocking accounting actions can result in faster aggregate scheduling and better client response times, though individual actions can, in some cases, be shortly delayed. By default, Moab uses non-blocking calls for the final charge *only*. The default behavior for individual accounting actions (such as Create, Start, End) can be overridden via the associated parameter (`CONTINUEISBLOCKING`, `CREATEISBLOCKING`, `DELETEISBLOCKING`, `ENDISBLOCKING`, `PAUSEISBLOCKING`, `RESUMEISBLOCKING`, `STARTISBLOCKING`).

For best performance when using non-blocking accounting actions, we recommend specifying a resource manager poll interval with a minimum poll time of zero (such as `RMPOLLINTERVAL=0,30`). Setting a non-zero minimum poll time can prevent Moab from responding quickly to accounting actions and can result in increased latency in job scheduling.

When using the `fast-allocation` accounting mode, if the charge action is set to be non-blocking (which is the default), Moab's account balance cache is not updated with the effects of the charge until the iteration after the charge is issued.

5.5.10 Retrying Failed Charges

If the AMCFG[] RETRYFAILEDCHARGES parameter is set to true (the default), job charges will be retried if they have failed due to a connection failure. When a job charge or usage record update (such as might occur when a job is suspended, at the periodic charge interval, or when a job completes) results in a connection failure between Moab and the accounting manager, then the charge request will be saved to a file in `SPOOLDIR/am/retrying/`. Once Moab detects that the connection with the accounting manager has been restored, the charge will be retried up to `CHARGERETRYCOUNT` times.

Charges that fail due to reasons other than a connection failure, or connection failures that surpass the CHARGERETRYCOUNT, will be saved to files in `SPOOLDIR/am/failed/`. Although these failures generally represent permanent failures, in some cases it might be possible to reissue some of these charges with a slight modification. For example, a user might have been moved from one account to another after the job started causing the final charge to fail. For such circumstances, a script has been provided (TOOLSDIR/mam/mam-charge-retry.pl) to facilitate the re-issuance of a failed usage charge from a failed charge retry file.

```
[root]# /software/moab-accounting/tools/mam/mam-charge-retry.pl --help
```

The mam-charge-retry.pl script mimics the mam-charge command in making a charge to MAM. The specified command-line options will override the original values contained in the failed charge file. The --dry-run option can be used to issue the retry as a quote rather than a charge in order to see if the charge would be successful. The --delete-on-success option can be used to delete the retry file after a successful charge. This script cannot be used to rerun a command when the accounting action uses a Native script. In such cases, the modified request XML from the charge retry file can be passed as the standard input to the Native script to reissue a charge.

Using the Script

This section provides synopsis information and an example on using the TOOLSDIR/mam/mam-charge-retry.pl script.

Synopsis

mam-charge-retry {[--filename] `<retry_filename>`} [-J `<instance_name>`] [-j `<usage_record_id>`] [-q `<quote_id>`] [-l `<lien_id>`] [-T `<usage_record_type>`] [-u `<user_name>`] [-g `<group_name>`] [-a `<account_name>`] [-o `<organization_name>`] [-c `<class_name>`] [-Q `<quality_of_service>`] [-m `<machine_name>`] [-N `<nodes>`] [-P `<processors>`] [-C `<cpu_time>`] [-M `<memory>`] [-D `<disk>`] [--stage `<lifecycle_stage>`] [-X, --extension `<property>=<value>`]... [-t `<charge_duration>`] [-s `<charge_start_time>`] [-e `<charge_end_time>`] [-d `<charge_description>`] [-z `<charge_amount>`]] [-f `<fund_id>`]] [--incremental] [-R `<charge_rate_name>`][{`<charge_rate_value>`}]=`<charge_rate_amount>`],...]... [--hours] [--itemize] [--delete-on-success] [--dry-run] [--debug] [--site `<site_name>`]] [--help] [--man] [--quiet] [--verbose] [--version]

Reissuing a Charge that has Failed Example

First we will list the files in the `SPOOLDIR/am/failed` directory to see if there are any 'permanently' failed charges that we might want to reissue:

```
[root]# cd /opt/moab/spool/am/failed
[root]# ls
job.250
```

We see there is a failed charge for job 250. It might be useful to check the charge file and examine the message to see what went wrong.

```
[root]# cat job.250
{"action":"End","message":"Failure registering job End (250) with accounting manager -
- Unable to invoke AM request - server rejected request with status code 740 - Failed
charging 1.00 credits for instance 250 and created usage record
25\nUser amy is not a valid member of Account biology","request":"<Request
action=\"Charge\"><Object>UsageRecord</Object><Option name=\"AccountingMode\">strict-
allocation</Option><Option name=\"StartTime\">1432070300</Option><Option
name=\"Duration\">300</Option><Data><UsageRecord><Stage>End</Stage><Type>Job</Type><In
stance>
250</Instance><User>amy</User><Group>staff</Group><Account>biology</Account><Class>bat
ch</Class><QualityOfService>premium</QualityOfService>
<Machine>colony</Machine><Nodes>1</Nodes><Partition>colony</Partition><Processors
consumptionRate=\"1.00\">12</Processors><StartTime>1432070300</StartTime><SubmitTime>1
432070300</SubmitTime><EndTime>1432070600</EndTime>
<ExitCode>0</ExitCode></UsageRecord></Data></Request>"}
```

We can see that this charge failed because the user (amy) was not a member of the specified account (biology). In this case, the user was a member of the biology account when the job started, but had been moved to the account chemistry by the time the job ended, resulting in a charge failure.

If we were to reissue the charge without modification, it would fail again, as we can see by using the script with the --dry-run option:

```
[root]# /opt/moab/tools/mam/mam-charge-retry.pl job.250 --dry-run
User amy is not a valid member of Account biology
```

We can reissue the charge after changing the request to use her new chemistry account:

```
[root]# /opt/moab/tools/mam/mam-charge-retry.pl job.250 -a chemistry --dry-run
Successfully quoted 1.00 credits for instance 250
```

Since that looks like it will work correctly, we'll issue the corrected charge request and delete the charge file:

```
[root]# /opt/moab/tools/mam/mam-charge-retry.pl job.250 -a chemistry --delete-on-
success
Successfully charged 1.00 credits for instance 250 and created usage record 35
```

Related Topics

- 5.6 AMCFG Parameters and Flags
- 2.7 Credentials - Per Class DISABLEAM attribute
- ENFORCEACCOUNTACCESS parameter

5.6 AMCFG Parameters and Flags

In this section:

5.6.1 AMCFG Parameters

Moab's accounting manager policies are defined using the AMCFG[] parameter. All AMCFG parameters must use the same accounting manager name between the square brackets (e.g., AMCFG[mam]).

The following AMCFG parameter values are supported:

ALWAYSCHARGERESERVATIONS	ENDISBLOCKING	QUERYURL
BACKUPHOST	ENDURL	REFRESHPERIOD
BLOCKINGACTIONS	FALLBACKACCOUNT	RESERVATIONCHARGEPOLICY
CHARGEPOLICY	FALLBACKQOS	RESUMEFAILUREACTION
CHARGERETRYCOUNT	FLAGS	RESUMEISBLOCKING
CONTINUEFAILUREACTION	FLUSHINTERVAL	RESUMEURL
CONTINUEISBLOCKING	HOST	RETRYFAILEDCHARGES
CONTINUEURL	INCLUDEFEATURES	SERVER
CREATECRED	JOBCHARGEPOLICY	STARTFAILUREACTION
CREATEFAILUREACTION	LIENGRANULARITY	STARTISBLOCKING
CREATEISBLOCKING	MODE	STARTURL
CREATEURL	NODECHARGEPOLICY	TIMEOUT
DELETEISBLOCKING	PAUSEISBLOCKING	TYPE
DELETEURL	PAUSEURL	UPDATEURL
DISABLEDACTIONS	PORT	VALIDATEJOBSUBMISSION

ALWAYSCHARGERESERVATIONS	
Description	This parameter is deprecated and may be removed in a future release. Use RESERVATIONCHARGEPOLICY instead. • If you were using AMCFG[] ALWAYSCHARGERESERVATIONS=True, comparable functionality can be obtained by using the new AMCFG[]

ALWAYSCHARGERESERVATIONS	
	RESERVATIONCHARGEPOLICY=All. • If you were using AMCFG[] ALWAYSCHARGERESERVATIONS=False, comparable functionality can be obtained by using the new AMCFG[] RESERVATIONCHARGEPOLICY=Select.

BACKUPHOST	
Format	STRING
Default	---
Description	The backup host name for the accounting manager server daemon.
Example	`AMCFG[mam] BACKUPHOST=headnode2` *Use the backup accounting manager server on headnode2 if the connection fails to the primary accounting manager server.*

BLOCKINGACTIONS	
Description	This parameter is deprecated and may be removed in a future release. Instead, specify the corresponding AMCFG[] CREATEISBLOCKING, DELETEISBLOCKING, ENDISBLOCKING, PAUSEISBLOCKING, RESUMEISBLOCKING, and STARTISBLOCKING parameters.

CHARGERETRYCOUNT	
Format	<INTEGER> (non-negative)
Default	`24`
Description	Only applicable if RETRYFAILEDCHARGES is enabled. The maximum number of times that Moab will retry failed connection charges. Moab will continue to retry until the charge succeeds, the charge fails due to a non-connection failure, or until the `CHARGERETRYCOUNT` limit is reached. If set to zero, no retries will be performed, and all charge failures will be written to files in the `SPOOL/am/failed/` directory.

CHARGERETRYCOUNT	
Example	`AMCFG[mam] RETRYFAILEDCHARGES=TRUE CHARGERETRYCOUNT=12` *Moab will retry connection-oriented charge failures up to 12 times.*

CHARGEPOLICY	
Description	This parameter is deprecated and may be removed in a future release. The policy guiding whether to charge for all jobs or just successful jobs has been subsumed into the new AMCFG[] JOBCHARGEPOLICY parameter. Rather than specifying in Moab whether you want to charge for allocated processors, blocked processors, processor equivalents, or CPU time, Moab *now* sends all of these resource usage properties to the accounting manager so that they can be tracked and charged independently.

CONTINUEFAILUREACTION	
Format	<GeneralFailureAction>[,<FundsFailureAction>[,<ConnectionFailureAction>]] where the action is one of `CANCEL` or `IGNORE`
Default	`IGNORE,IGNORE,IGNORE`
Description	Setting this parameter to `IGNORE` can result in under-charging, and possibly over-charging of jobs. We recommend using `DEFER,CANCEL,DEFER` in most cases, unless it is acceptable to have occasional errors in the job charges. If periodic charging is enabled (via the AMCFG[] FLUSHINTERVAL parameter), this parameter specifies the action to be taken if a failure is detected when Moab performs its periodic accounting update (e.g., to determine whether the job should be continued): • Moab applies <ConnectionFailureAction> to a job if it is rejected due to a connection failure to MAM. • Moab applies <FundsFailureAction> to a job if it is rejected due to insufficient funds. • Moab applies <GeneralFailureAction> to a job if the accounting manager rejects it for any other reason. • If you do not specify a <ConnectionFailureAction>, or if you do not

CONTINUEFAILUREACTION	
	specify a <FundsFailureAction>, then Moab will apply the <GeneralFailureAction> for the unspecified case. If the action is set to `CANCEL`, Moab cancels the job; for `IGNORE`, Moab ignores the failure and continues running the job.
Example	`AMCFG[mam] CONTINUEFAILUREACTION=IGNORE,CANCEL,IGNORE` *A job will be canceled if there are insufficient funds when Moab performs its periodic accounting update; but will be allowed to continue running if MAM is down, or for any other reason.*

CONTINUEISBLOCKING	
Format	<BOOLEAN>
Default	`TRUE`
Description	If set to `TRUE`, the scheduler will block while authorizing the continuation of a job with the accounting manager. If set to `FALSE`, the accounting operation will be queued to the accounting thread pool and scheduling will continue; but application of the failure action will be delayed until a response is received.
Example	`AMCFG[mam] CONTINUEISBLOCKING=FALSE` *Specifies that Moab should use non-blocking calls with the accounting manager when checking to see if a job should be continued after a periodic accounting update.*

CONTINUEURL	
Format	exec://<fullPathToContinueScript> or null:
Default	exec://$TOOLSDIR/mam/usage.reserve.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	If periodic charging is enabled (via AMCFG[] FLUSHINTERVAL), when Moab performs a periodic accounting update for a job, this script is invoked to determine whether there are sufficient allocations for it to continue running for another period. For jobs, the CONTINUEFAILUREACTION attribute specifies the action that Moab should take if the authorization fails (such as for insufficient funds). If

CONTINUEURL	
	you use a job charge policy of Successful, Moab will not call the script because it does not yet know the completion status of the job. To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] CONTINUEURL=exec://$TOOLSDIR/mam/usage.continue.custom.pl` *Moab calls the usage.continue.custom.pl script for authorization when checking to see if a job should be continued after a periodic accounting update.*

CREATECRED	
Format	<BOOLEAN>
Default	FALSE
Description	If set to TRUE, Moab will be enabled to query accounts, users, user membership in accounts, and users' default accounts from Moab Accounting Manager and define them in Moab. These credentials can be manually updated by running `mrmctl -R AM` or automatically updated be setting the AMCFG[] REFRESHPERIOD parameter. If you want Moab to enforce the imported account-user memberships, you will need to set the ENFORCEACCOUNTACCESS parameter to TRUE. See Appendix A: Moab Parameters for more information on the ENFORCEACCOUNTACCESS parameter.
Example	`AMCFG[mam] CREATECRED=TRUE REFRESHPERIOD=30:00` *Moab will automatically update account credential information from MAM every half hour.*

CREATEFAILUREACTION	
Format	<GeneralFailureAction>[,<FundsFailureAction>[,<ConnectionFailureAction>]] where the action is one of CANCEL, DEFER, HOLD, or IGNORE
Default	IGNORE,IGNORE,IGNORE
Description	Before creating a job that should be tracked or charged within the accounting manager, Moab contacts the accounting manager for authorization:

CREATEFAILUREACTION	
	• If the job creation is rejected due to a connection failure with MAM, Moab applies the <ConnectionFailureAction> to the job. • If the job creation is rejected due to lack of funds, Moab applies the <FundsFailureAction> to the job. • For any other rejection reason, Moab applies the <GeneralFailureAction> to the job. • If you do not specify a <ConnectionFailureAction>, or if you do not specify a <FundsFailureAction>, then Moab will apply the <GeneralFailureAction> for the unspecified case. If the action is set to `CANCEL`, Moab cancels the job; `DEFER`, Moab defers the job; `HOLD`, Moab puts the job on hold; `IGNORE`, Moab ignores the failure and continues to start the job. In order for the `CREATEFAILUREACTION` policy to be applied, the AMCFG[] `VALIDATEJOBSUBMISSION` parameter must be set to true. If you have FALLBACKQOS or FALLBACKACCOUNT defined and a user requests an account with insufficient funds, the job will still be moved to the fallback credential, regardless of the action defined for `CREATEFAILUREACTION`.
Example	`AMCFG[mam] CREATEFAILUREACTION=CANCEL,HOLD,IGNORE` *A job will be placed on hold when submitted if there are insufficient funds for it to start. However, it will be allowed to be submitted if there is a connection problem with MAM. The job will be canceled if there is any other failure (e.g., the user does not belong to the specified account).*

CREATEISBLOCKING	
Format	<BOOLEAN>
Default	`TRUE`
Description	If set to `TRUE`, the scheduler will block while authorizing the creation of a job with the accounting manager. If set to `FALSE`, the accounting operation will be queued to the accounting thread pool and scheduling will continue, but further consideration for the job will be delayed until a response is received.
Example	`AMCFG[mam] CREATEISBLOCKING=FALSE`

CREATEISBLOCKING	
	Specifies that Moab should use non-blocking calls with the accounting manager when creating jobs.

CREATEURL	
Format	exec://<fullPathToCreateScript> or null:
Default	exec://$TOOLSDIR/mam/usage.quote.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	Moab runs this script at the time a job or reservation is being created. For jobs, the `CREATEFAILUREACTION` attribute specifies the action that should be taken if the authorization fails (such as for insufficient funds). To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] CREATEURL=exec://$TOOLSDIR/mam/usage.create.custom.pl` *Moab calls the `usage.create.custom.pl` script for authorization before starting a job or reservation.*

DELETEISBLOCKING	
Format	<BOOLEAN>
Default	`TRUE`
Description	If set to `TRUE`, the scheduler will block while contacting the accounting manager to clean up after a failed job start. If set to `FALSE`, the accounting operation will be queued to the accounting thread pool and scheduling will continue.
Example	`AMCFG[mam] DELETEISBLOCKING=FALSE` *Specifies that Moab should use non-blocking calls with the accounting manager when cleaning up after failed job starts.*

DELETEURL	
Format	exec://<fullPathToDeleteScript> or null:
Default	exec://$TOOLSDIR/mam/lien.delete.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	Moab runs this script to clean up after an interrupted job or reservation life-cycle. The default behavior is to remove outstanding liens. To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] DELETEURL=exec://$TOOLSDIR/mam/usage.delete.custom.pl` *Moab calls the* `usage.delete.custom.pl` *script to clean up after an interrupted job or reservation.*

DISABLEDACTIONS	
Description	This parameter is deprecated and may be removed in a future release. Instead, specify an empty value or a protocol of 'null:' for the corresponding AMCFG[] CREATEURL, DELETEURL, ENDURL, PAUSEURL, RESUMEURL, STARTURL, and UPDATEURL parameters.

ENDISBLOCKING	
Format	<BOOLEAN>
Default	`FALSE`
Description	If set to `TRUE`, the scheduler will block while registering the end of a job with the accounting manager. If set to `FALSE`, the accounting operation will be queued to the accounting thread pool and scheduling will continue.
Example	`AMCFG[mam] ENDISBLOCKING=FALSE` *Specifies that Moab should use non-blocking calls with the accounting manager when a job ends.*

ENDURL	
Format	exec://<fullPathToEndScript> or null:
Default	exec://$TOOLSDIR/mam/usage.charge.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	Moab runs this script after the end of a chargeable job or reservation in order to make a final charge or update the accounting record. The default behavior is to make a prorated charge for the job or reservation. To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] ENDURL=exec://$TOOLSDIR/mam/usage.end.custom.pl` *Moab calls the `usage.end.custom.pl` script to make the final charge for a job or reservation.*

FALLBACKACCOUNT	
Format	<STRING>
Default	---
Description	If specified, Moab verifies adequate allocations for all new jobs. If adequate allocations are not available in the job's primary account, Moab changes the job's credentials to use the fallback account. If not specified, Moab places a hold on jobs that do not have adequate allocations in their primary account.
Example	`AMCFG[mam] FALLBACKACCOUNT=freecycle` *Moab assigns the account `freecycle` to jobs that do not have adequate allocations in their primary account.*

FALLBACKQOS	
Format	<STRING>
Default	---
Description	If specified, Moab verifies adequate allocations for all new jobs. If adequate allocations are not available in the job's primary QoS, Moab changes the job's credentials to use the fallback QoS. If not specified, Moab places a hold on jobs

FALLBACKQOS	
	that do not have adequate allocations in their primary QoS.
Example	`AMCFG[mam] FALLBACKQOS=freecycle` *Moab assigns the QoS `freecycle` to jobs that do not have adequate allocations in their primary QoS.*

FLAGS	
Format	<STRING>
Default	---
Description	`AMCFG` flags are used to enable special services.
Example	`AMCFG[mam] FLAGS=LOCALCOST` *Moab calculates the charge for the job locally and sends that as a charge to the accounting manager, which then charges that amount for the job.* This LOCALCOST flag is deprecated and may be removed in a future release.

FLUSHINTERVAL	
Format	[[`DD:`]`HH:`]`MM:`]`SS` or `INFINITY` The former values of HOUR, DAY, WEEK, MONTH, or NONE are deprecated and may be removed in a future release.
Default	`INFINITY`
Description	Indicates the amount of time between accounting manager updates for long running reservations and jobs. If `FLUSHINTERVAL` is set to a positive time period, Moab will update the accounting manager (e.g., make an incremental charge) on the specified period relative to the start of the job or reservation. If `FLUSHINTERVAL` is set to `INFINITY`, the update will only occur at the end of the job or reservation.

FLUSHINTERVAL	
Example	`AMCFG[mam] FLUSHINTERVAL=1:00:00:00` *Moab will make periodic accounting updates every 24 hours for long running jobs and reservations.*

HOST	
Format	<STRING>
Default	`localhost`
Description	The host name for the accounting manager server daemon.
Example	`AMCFG[mam] HOST=my-mam-server` *Moab will communicate with the MAM server running on `my-mam-server`.*

INCLUDEFEATURES	
Format	One of `NodeCombination`, `NodeCount`, `TaskCombination`, `TaskCount`, or `NONE`
Default	`NONE`
Description	Moab will pass allocated node features to the accounting manager under the Features property according to the specified policy: • If set to `NodeCombination`, Moab will pass a list of aggregate combinations of features for each node in the job. • If set to `NodeCount`, Moab will pass a list of features counted over each node in the job. • If set to `TaskCombination`, Moab will pass a list of aggregate combinations of features for each node in each task in the job. • If set to `TaskCount`, Moab will pass a list of features counted over each node in each task in the job.
Example	For example, assuming a job that runs with two tasks on each of eight nodes (e.g., -l nodes=8:ppn=2); four nodes; two having a single bigmem feature, two having a single fastcpu feature, two having both the bigmem and the fastcpu feature, and two having no features. The following parameter values pass the accompanying Features value to the accounting manager:

INCLUDEFEATURES	

	`AMCFG[mam] IncludeFeatures=NodeCombination Features=bigmem:2,bigmem+fastcpu:2,fastcpu:2,None:2`
	`AMCFG[mam] IncludeFeatures=NodeCount Features=bigmem:4,fastcpu:4`
	`AMCFG[mam] IncludeFeatures=TaskCombination Features=bigmem:4,bigmem+fastcpu:4,fastcpu:4,None:4`
	`AMCFG[mam] IncludeFeatures=TaskCount Features=bigmem:8,fastcpu:8`

JOBCHARGEPOLICY	
Format	One of `All`, `None`, or `Successful`
Default	`All`
Description	Specifies whether all, successful only, or no jobs should be charged for their resource usage: • If set to `All` (the default), all jobs are charged for their resource usage, independent of their completion status. • If set to `None`, no jobs will be tracked or charged for their resource usage. • If set to `Successful`, only jobs having a successful exit status (0) are charged for their resource usage. If the LOCALCOST flag (AMCFG[] FLAGS=LOCALCOST) is set, Moab overrides the charge sent to the accounting manager and instead charges a value of the processors times duration (additionally factoring in NODECHARGE is defined). If LOCALCOST is *not* set, Moab allows the accounting manager to calculate the job charges. This LOCALCOST flag is deprecated and may be removed in a future release.
Example	`AMCFG[mam] JOBCHARGEPOLICY=Successful` *Charge only for jobs having a successful exit status.*

LIENGRANULARITY	
Format	One of `Partial` or `Combined`

LIENGRANULARITY	
Default	`Partial`
Description	When periodic charging is enabled via AMCFG[] FLUSHINTERVAL, lien granularity controls whether a combined lien is sought for the duration of the entire job (Combined) or whether partial liens are sought for the duration of each periodic charge interval (Partial): • When using a lien granularity of Partial, a job or reservation may get started if it has enough funds to run for the `FLUSHINTERVAL`, but it might trigger a `CONTINUEFAILUREACTION` if it runs out of funds before completion. • When using a lien granularity of Combined, the funds for the entire job or reservation must be available before it starts, but the funds will be protected by the lien and consumed on a periodic interval.
Example	`AMCFG[mam] LIENGRANULARITY=Combined` *When using periodic charging, Moab will seek to obtain a lien for the entire duration of the job or reservation before starting it.*

MODE	
Format	One of `strict-allocation`, `fast-allocation`, `notional-charging`, or `usage-tracking`
Default	`strict-allocation`
Description	The accounting mode. The accounting mode modifies the way in which accounting-relevant job stages (e.g., create, start, end, etc.) are processed. See 5.5.2 Accounting Mode for details on the behavior of the accounting modes.
Example	`AMCFG[mam] MODE=notional-charging` *Configures Moab to use the notional-charging accounting mode when interacting with the accounting manager.*

NODECHARGEPOLICY	
Format	One of `AVG`, `MAX`, or `MIN`
Default	`MIN`

NODECHARGEPOLICY

Description

When charging for resource usage, the accounting manager will charge by node allocation according to the specified policy. For AVG, MAX, and MIN, the accounting manager will charge by the average, maximum, and minimum node charge rate of all allocated nodes.

If you use this feature in conjunction with the AMCFG[] LOCALCOST flag, Moab will include the calculation of the node charge value sent to MAM. See the LOCALCOST flag below.

This LOCALCOST flag is deprecated and may be removed in a future release.

If you do not use this feature in conjunction with the AMCFG[] LOCALCOST flag, you must perform the following MAM commands to include node charges in charge calculations:

- Add NodeCharge as a usage record property:

```
mam-shell Attribute Create Object=UsageRecord Name=NodeCharge
DataType=Float Description="\"Node Charge\""
```

- Add NodeCharge as a multiplier charge rate:

```
mam-create-chargerate -n NodeCharge -z "*1" -d "Node Charge Multiplier"
```

Example

```
NODECFG[node01]  CHARGERATE=1.5
NODECFG[node02]  CHARGERATE=1.75
AMCFG[mam] NODECHARGEPOLICY=MAX
```

Charge jobs by the maximum allocated node's charge rate.

PAUSEISBLOCKING

Format	<BOOLEAN>
Default	TRUE
Description	If set to TRUE, the scheduler will block while registering the suspension of a job with the accounting manager. If set to FALSE, the accounting operation will be queued to the accounting thread pool and scheduling will continue.

Example

```
AMCFG[mam] PAUSEISBLOCKING=FALSE
```

Specifies that Moab should use non-blocking calls with the accounting manager when suspending jobs.

PAUSEURL	
Format	exec://<fullPathToPauseScript> or null:
Default	exec://$TOOLSDIR/mam/usage.charge.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	Moab runs this script after preempting a job that might be resumed later. The default behavior is to make an incremental charge but not create a fresh lien. If you use a job charge policy of Successful, Moab will not call the script because it does not yet know the completion status of the job. To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] PAUSEURL=exec://$TOOLSDIR/mam/usage.pause.custom.pl` *Moab calls the `usage.pause.custom.pl` script after pausing a job.*

PORT	
Format	<INTEGER>
Default	7112
Description	The listening port for the accounting manager server daemon.
Example	`AMCFG[mam] PORT=7731` *Moab will communicate with the MAM server listening on port `7731`.*

QUERYURL	
Format	exec://<fullPathToQueryScript> or null:
Default	exec://$TOOLSDIR/mam/account.query.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	Moab runs this script to customize and forward the Moab query to the accounting manager. The standard input to the script will be an XML Request in SSS format and is used directly between Moab and Moab Accounting Manager. Its primary purpose is to synchronize accounts and user information with the

QUERYURL	
	accounting manager if the CREATECRED parameter is specified. To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] QUERYURL=exec://$TOOLSDIR/mam/cred.query.custom.pl` *Moab calls the `cred.query.custom.pl` script in order to obtain account and user information from the accounting manager.*

REFRESHPERIOD	
Format	[[[`DD:`]`HH:`]`MM:`]`SS` or `INFINITY` The former values of HOUR, DAY, WEEK, MONTH, or NONE are deprecated and may be removed in a future release.
Default	`INFINITY`
Description	Indicates the period at which Moab will poll for updated information from Moab Accounting Manager (MAM): • If AMCFG[] CREATECRED is set to `TRUE`, Moab will update the accounting credentials from MAM on the specified period. • If AMCFG[] MODE is set to `fast-allocation`, Moab will update the account balance cache from MAM on the specified period. Moab will poll MAM for updated information when it first starts up unless REFRESHPERIOD is set to `0`. If REFRESHPERIOD is set to a positive time period, Moab will refresh the accounting credentials on the specified period relative to the scheduler start time. If REFRESHPERIOD is set to `INFINITY`, Moab will only request updated information from MAM when first started. Use *`mrmctl -R am`* to force an immediate refresh.
Example	`AMCFG[mam] REFRESHPERIOD=2:00:00` *Moab will request an update from MAM every two hours.*

RESERVATIONCHARGEPOLICY	
Format	One of `All`, `None`, or `Select`

RESERVATIONCHARGEPOLICY	
Default	`Select`
Description	If set to `All`, idle cycles in reservations will be charged to the accounting manager by default, even if the ChargeAccount and ChargeUser are not specified for the reservation. For reservations that you do not want to be charged with the accounting manager, specify the reservation Charge attribute with a value of False. If set to `None`, idle cycles in reservations will never be charged to the accounting manager, even if you have specified the ChargeAccount, ChargeUser or the Charge attribute with a value of True. If set to `Select` (the default), idle cycles in reservations will not be charged to the accounting manager unless you specify the reservation ChargeAccount or ChargeUser attributes or set the reservation Charge attribute with a value of True.
Example	`AMCFG[mam] RESERVATIONCHARGEPOLICY=All` *By default, Moab will charge for idle cycles in reservations unless overridden with Charge=False.*

RESUMEFAILUREACTION	
Format	<GeneralFailureAction>[,<FundsFailureAction>[,<ConnectionFailureAction>]] where the action is one of `CANCEL`, `DEFER`, `HOLD`, `IGNORE`, or `RETRY`
Default	`IGNORE,IGNORE,IGNORE`
Description	Setting this parameter to `IGNORE` can result in under-charging, and possibly over-charging of jobs. We recommend using `DEFER,CANCEL,DEFER` in most cases, unless it is acceptable to have occasional errors in the job charges. This action is applied after a failure with the accounting manager when a job is being resumed (e.g., after being suspended): • Moab will apply <ConnectionFailureAction> to a job if there is a connection failure between Moab and the accounting manager. • Moab applies <FundsFailureAction> to the job if it is rejected due to insufficient funds. • Moab applies <GeneralFailureAction> to a job if the accounting manager rejects it for any other reason. • If you do not specify a <ConnectionFailureAction>, or if you do not specify a <FundsFailureAction>, then Moab will apply the

RESUMEFAILUREACTION	
	<GeneralFailureAction> for the unspecified case. If the action is set to `CANCEL`, Moab cancels the job; `DEFER`, Moab defers the job; `HOLD`, Moab puts the job on hold; `IGNORE`, Moab ignores the failure and continues to resume the job; `RETRY`, Moab does not resume the job on this attempt but will continue to try to resume the job at the next opportunity.
Example	`AMCFG[mam] RESUMEFAILUREACTION=HOLD,HOLD,IGNORE` *A job will be resumed if Moab is unable to contact the accounting manager. Otherwise, the job will be placed on hold if there is any other failure with the accounting manager when Moab tries to resume it.*

RESUMEISBLOCKING	
Format	<BOOLEAN>
Default	`TRUE`
Description	If set to `TRUE`, the scheduler will block while authorizing the resumption of a job with the accounting manager. If set to `FALSE`, the accounting operation will be queued to the accounting thread pool and scheduling will continue, but resumption of the job will be delayed until a response is received.
Example	`AMCFG[mam] RESUMEISBLOCKING=FALSE` *Specifies that Moab should use non-blocking calls with the accounting manager when resuming jobs.*

RESUMEURL	
Format	exec://<fullPathToResumeScript> or null:
Default	exec://$TOOLSDIR/mam/usage.reserve.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	Moab runs this script before resuming a suspended job to determine whether there it has authorization to resume (e.g., has sufficient funds). For jobs, the `RESUMEFAILUREACTION` attribute specifies the action that Moab should take if the authorization fails (such as for insufficient funds). If you use a job charge policy of Successful, Moab will not call the script because it

RESUMEURL	
	does not yet know the completion status of the job. To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] RESUMEURL=exec://$TOOLSDIR/mam/usage.resume.custom.pl` *Moab calls the* `usage.resume.custom.pl` *script for authorization before resuming a suspended job.*

RETRYFAILEDCHARGES	
Format	<BOOLEAN>
Default	`TRUE`
Description	If set to `TRUE`, job charges will be retried if they have failed due to a connection failure. When a job charge or usage record update (such as might occur when a job is suspended, at the periodic charge interval, or when a job completes) results in a connection failure between Moab and the accounting manager, then the charge request will be saved to a file in `SPOOLDIR/am/retrying/`. Once Moab detects that the connection with the accounting manager has been restored, the charge will be retried up to `CHARGERETRYCOUNT` times. Charges that fail due to reasons other than a connection failure, or connection failures that surpass the `CHARGERETRYCOUNT`, will be saved to files in `SPOOLDIR/am/failed/`.
Example	`AMCFG[mam] RETRYFAILEDCHARGES=TRUE` *Moab will retry connection-oriented charge failures.*

SERVER	
Format	<URL>
Default	N/A
Description	The type and location of the accounting manager service.
Example	`AMCFG[mam] SERVER=mam://tiny.supercluster.org:4368`

STARTFAILUREACTION	
Format	<GeneralFailureAction>[,<FundsFailureAction>[,<ConnectionFailureAction>]] where the action is one of `CANCEL`, `DEFER`, `HOLD`, `IGNORE`, or `RETRY`
Default	`IGNORE,IGNORE,IGNORE`
Description	Setting this parameter to `IGNORE` can result in under-charging, and possibly over-charging of jobs. We recommend using `DEFER,CANCEL,DEFER` in most cases, unless it is acceptable to have occasional errors in the job charges. Moab applies the appropriate failure action if there is a failure when registering the job start with the accounting manager: • Moab applies <ConnectionFailureAction> to the job if there is a communication problem with the accounting manager. • Moab applies <FundsFailureAction> to the job if it is rejected due to insufficient funds. • Moab applies <GeneralFailureAction> to a job if the accounting manager rejects it for any other reason. • If you do not specify a <ConnectionFailureAction>, or if you do not specify a <FundsFailureAction>, then Moab will apply the <GeneralFailureAction> for the unspecified case. If the action is set to `CANCEL`, Moab cancels the job; `DEFER`, Moab defers the job; `HOLD`, Moab puts the job on hold; `IGNORE`, Moab ignores the failure and continues to start the job; and `RETRY`, Moab does not start the job on this attempt but attempts to start the job at the next opportunity.
Example	`AMCFG[mam] STARTFAILUREACTION=CANCEL,HOLD,IGNORE` *A job will be placed on hold if there are insufficient funds when it is time for it to start. It will be allowed to start if Moab is unable to reach the accounting manager. For all other failures with the accounting manager, the job will be canceled.*

STARTISBLOCKING	
Format	<BOOLEAN>
Default	`TRUE`
Description	If set to `TRUE`, the scheduler will block while authorizing the starting of a job with the accounting manager. If set to `FALSE`, the accounting operation will be

STARTISBLOCKING	
	queued to the accounting thread pool and scheduling will continue, but the start of the job will be delayed until a response is received. ⚠ If using Moab in a Peer-to-Peer grid, do not set this parameter to `FALSE`. The Start action is not supported as a non-blocking action in Peer-to-Peer grids.
Example	`AMCFG[mam] STARTISBLOCKING=FALSE` *Specifies that Moab should use non-blocking calls with the accounting manager when starting jobs.*

STARTURL	
Format	exec://<fullPathToStartScript> or null:
Default	exec://$TOOLSDIR/mam/usage.reserve.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	Moab runs this script on a chargeable job or reservation to determine whether it should start. For jobs, the `STARTFAILUREACTION` attribute specifies the action that Moab should take if the authorization fails (such as for insufficient funds). To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] STARTURL=exec://$TOOLSDIR/mam/usage.start.custom.pl` *Moab calls the `usage.start.custom.pl` script for authorization before starting a job or reservation.*

TIMEOUT	
Format	[[[DD:]HH:]MM:]SS
Default	15
Description	The maximum delay allowed for communications with the accounting manager.
Example	`AMCFG[mam] TIMEOUT=30`

TYPE	
Format	One of `MAM` or `Native`
Default	`MAM`
Description	The accounting manager interface type.
Example	`AMCFG[mam] TYPE=MAM` *Configures Moab to interact with MAM using the direct SSS wire protocol.*

UPDATEURL	
Format	exec://<fullPathToUpdateScript> or null:
Default	exec://$TOOLSDIR/mam/usage.charge.mam.pl if `TYPE=Native`; otherwise it will make a direct call to MAM (mam:)
Description	If you have `FLUSHINTERVAL` set, Moab runs this script every flush interval for each chargeable job or reservation to charge for the previous interval. This call is usually followed by a call to the `CONTINUEURL` script, if defined, to check whether there are sufficient funds to run for the next interval. If you use a job charge policy of Successful, Moab will not call the script because it does not yet know the completion status of the job. To disable a script from being run at this stage, use 'null:' as the parameter value.
Example	`AMCFG[mam] UPDATEURL=exec://$TOOLSDIR/mam/usage.update.custom.pl` *Moab calls the `usage.update.custom.pl` script for authorization to continue a job or reservation.*

VALIDATEJOBSUBMISSION	
Format	<BOOLEAN>
Default	`FALSE`
Description	If set to `TRUE`, when a new job is submitted, Moab will execute the `CREATEURL` script (for `TYPE=Native`) or seek a job quote from Moab Accounting Manager (`TYPE=MAM`) before allowing the job to be submitted.

VALIDATEJOBSUBMISSION	
	Otherwise, the fund validation step is just utilized by reservations and fallback account checks. If the call fails (for example, if the user's account does not have sufficient funds or specifies an invalid account), Moab applies the CREATEFAILUREACTION.
Example	`AMCFG[mam] VALIDATEJOBSUBMISSION=True CREATEFAILUREACTION=Hold` *Verify jobs have sufficient funds to run at the time they are submitted.*

5.6.2 AMCFG Flags

AMCFG flags can be used to enable special services and to disable default services. These services are enabled/disabled by setting the AMCFG FLAGS attribute (see the parameter FLAGS above).

Flag Name	Description
ACCOUNTFAILASFUNDS	When this flag is set, logic failures within the accounting manager are treated as fund failures and are canceled. When ACCOUNTFAILASFUNDS is not set, accounting manager failures are treated as a server failure and the result is a job that requests an account to which the user does not have access.
LOCALCOST	This flag is deprecated and may be removed in a future release.
STRICTQUOTE	Sends an estimated process count to the accounting manager when an initial quote is requested for a newly-submitted job.

Related Topics

- 5.5 Accounting, Charging, and Allocation Management

Chapter 6: Controlling Resource Access - Reservations, Partitions, and QoS Facilities

In this chapter:

6.1 Advance Reservations

An advance reservation is the mechanism by which Moab guarantees the availability of a set of resources at a particular time. Each reservation consists of three major components: (1) a set of resources, (2) a time frame, and (3) an access control list. It is a scheduler role to ensure that the access control list is not violated during the reservation's lifetime (that is, its time frame) on the resources listed. For example, a reservation may specify that node002 is reserved for user Tom on Friday. The scheduler is therefore constrained to make certain that only Tom's jobs can use node002 at any time on Friday. Advance reservation technology enables many features including backfill, deadline based scheduling, grid scheduling, and QOS support.

The mrsvctl command is used to create, modify, query, and release reservations.

In this section:

6.1.1 Reservation Overview

Every reservation consists of three major components: (1) a set of resources, (2) a time frame, and (3) an access control list. Additionally, a reservation can also have a number of optional attributes controlling its behavior and interaction with other aspects of scheduling. Reservation attribute descriptions follow.

In this topic:

6.1.1.A Resources

Under Moab, the resources specified for a reservation are specified by way of a task description. Conceptually, a task can be thought of as an atomic, or indivisible, collection of resources. If reservation resources are unspecified, a task is a node by default. To define a task, specify resources. The resources can include processors, memory, swap, local disk, and so forth. For example, a single task may consist of 1 processor, 2 GB of memory, and 10 GB of local disk.

A reservation consists of one or more tasks. In attempting to locate the resources required for a particular reservation, Moab examines all feasible resources and locates the needed resources in groups specified by the task description. An example may help clarify this concept:

Reservation A requires four tasks. Each task is defined as 1 processor and 1 GB of memory.

Node X has 2 processors and 3 GB of memory available
Node Y has 2 processors and 1 GB of memory available
Node Z has 2 processors and 2 GB of memory available

When collecting the resources needed for the reservation, Moab examines each node in turn. Moab finds that Node X can support 2 of the 4 tasks needed by reserving 2 processors and 2 GB of memory, leaving 1 GB of memory unreserved. Analysis of Node Y shows that it can only support 1 task reserving 1 processor and 1 GB of memory, leaving 1 processor unreserved. Note that the unreserved memory on Node X cannot be combined with the unreserved processor on Node Y to satisfy the needs of another task because a task requires all resources to be located on the same node. Finally, analysis finds that node Z can support 2 tasks, fully reserving all of its resources.

Both reservations and jobs use the concept of a task description in specifying how resources should be allocated. It is important to note that although a task description is used to allocate resources to a reservation, this description does not in any way constrain the use of those resources by a job. In the above example, a job requesting resources simply sees 4 processors and 4 GB of memory available in reservation A. If the job has access to the reserved resources and the resources meet the other requirements of the job, the job could use these resources according to its own task description and needs.

Currently, the resources that can be associated with reservations include processors, memory, swap, local disk, initiator classes, and any number of arbitrary resources. Arbitrary resources can include peripherals such as tape drives, software licenses, or any other site specific resource.

6.1.1.B Time Frame

Associated with each reservation is a time frame. This specifies when the resources will be reserved or dedicated to jobs that meet the reservation's access control list (ACL). The time frame simply consists of a start time and an end time. When configuring a reservation, this information can be specified as a start time together with either an end time or a duration.

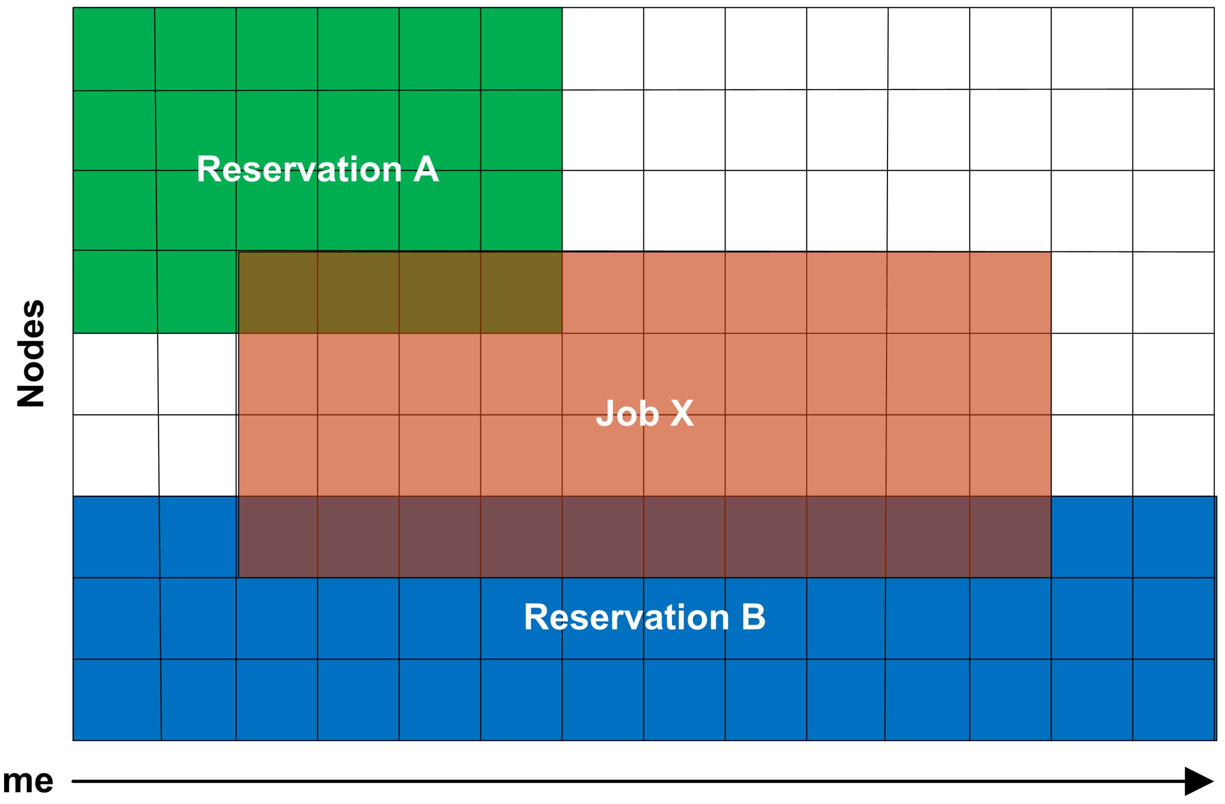
6.1.1.C Access Control List

A reservation's access control list specifies which jobs can use a reservation. Only jobs that meet one or more of a reservation's access criteria are allowed to use the reserved resources during the reservation time frame. Currently, the reservation access criteria include the following: users, groups, accounts, classes, QOS, job attributes, job duration, and job templates.

6.1.1.D Job to Reservation Binding

While a reservation's ACL will allow particular jobs to use reserved resources, it does not force any job to use these resources. With each job, Moab attempts to locate the best possible combination of available resources whether these are reserved or unreserved. For example, in the following figure, note that job `X`, which meets access criteria for both reservation `A` and `B`, allocates a portion of its resources from each reservation and the remainder from resources outside of both reservations.

Image 6-1: Job X uses resources from reservations A and B



Although by default, reservations make resources available to jobs that meet particular criteria, Moab can be configured to constrain jobs to only run within accessible reservations. This can be requested by the user on a job by job basis using a resource manager extension flag, or it can be enabled administratively via a QoS flag. For example, assume two reservations were created as follows:

```
> mrsvctl -c -a GROUP==staff -d 8:00:00 -h 'node[1-4]'
reservation staff.1 created
```

```
> mrsvctl -c -a USER==john -t 2
reservation john.2 created
```

If the user 'john,' who happened to also be a member of the group 'staff,' wanted to force a job to run within a particular reservation, 'john' could do so using the `FLAGS` resource manager extension. Specifically, in the case of a PBS job, the following submission forces the job to run within the 'staff.1' reservation:

```
> msub -l nodes=1,walltime=1:00:00,flags=ADVRES:staff.1 testjob.cmd
```

Note that for this to work, PBS needs to have resource manager extensions enabled as described in the PBS Resource Manager Extension Overview. (Torque has resource manager extensions enabled by default.) If the user wants the job to run on reserved resources but does not care which, the user could submit the job with the following:

```
> msub -l nodes=1,walltime=1:00:00,flags=ADVRES testjob.cmd
```

To enable job to reservation mapping via QoS, the QoS flag USERESERVED should be set in a similar manner.

Use the reservation BYNAME flag to require explicit binding for reservation access.

To lock jobs linked to a particular QoS into a reservation or reservation group, use the REQRID attribute.

6.1.1.E Reservation Specification

There are two main types of reservations that sites typically deal with. The first, administrative reservations, are typically one-time reservations created for special purposes and projects. These reservations are created using the mrsvctl or setres commands. These reservations provide an integrated mechanism to allow graceful management of unexpected system maintenance, temporary projects, and time critical demonstrations. This command allows an admin to select a particular set of resources or just specify the quantity of resources needed. For example an admin could use a regular expression to request a reservation be created on the nodes 'blue0[1-9]' or could simply request that the reservation locate the needed resources by specifying a quantity based request such as 'TASKS==20.'

The second type of reservation is called a standing reservation. It is specified using the SRCFG[X] parameter and is of use when there is a recurring need for a particular type of resource distribution. Standing reservations are a powerful, flexible, and efficient means for enabling persistent or periodic policies such as those often enabled using classes or queues. For example, you could use a standing reservation to reserve a subset of its compute resources for quick turnaround jobs during business hours on Monday through Friday. See 6.1.3 Standing Reservations for more information about configuring and using these reservations.

6.1.1.F Reservation Behavior

As previously mentioned, a given reservation can have one or more access criteria. A job can use the reserved resources if it meets at least one of these access criteria. It is possible to stack multiple reservations on the same node. In such a situation, a job can only use the given node if it has access to each active reservation on the node.

6.1.1.G Reservation Group

Reservations groups are ways of associating multiple reservations. This association is useful for variable namespace and reservation requests. The reservations in a group inherit the variables from the reservation group head, but if the same variable is set locally on a reservation in the group, the local variable overrides the inherited variable. Variable inheritance is useful for triggers as it provides greater flexibility with automating certain tasks and system behaviors.

Jobs can be bound to a reservation group (instead of a single reservation) by using the resource manager extension ADVRES.

6.1.1.H Infinite Jobs and Reservations

To allow infinite walltime jobs, you must have the following scheduler flag set:

```
SCHEDCFG[Moab] FLAGS=allowinfinitejobs
```

You can submit an infinite job by completing:

```
msub -l walltime=INFINITY
```

Or an infinite reservation by completing:

```
mrsvctl -c -d INFINITY
```

Infinite jobs can run in infinite reservations. Infinite walltime also works with job templates and advres.

Output XML for infinite jobs will print 'INFINITY' in the ReqAWDuration, and XML for infinite rsvs will print 'INFINITY' in duration and endtime:

```
<Data>
   <rsv AUser="jgardner" AllocNodeCount="1" AllocNodeList="n5"
   AllocProcCount="4" AllocTaskCount="1" HostExp="n5"
   LastChargeTime="0" Name="jgardner.1" Partition="base"
   ReqNodeList="n5:1" Resources="PROCS=[ALL]" StatCBPS="0"
   StatCRPS="800" StatTBPS="0" StatTRPS="0" SubType="Other"
   Type="User" cost="0.000000" ctime="1302127058"
   duration="INFINITY" endtime="INFINITY" starttime="1302127058">
     <ACL aff="neutral" cmp="%=" name="jgardner.1" type="RSV"></ACL>
     <ACL cmp="%=" name="jgardner" type="USER"></ACL>
     <ACL cmp="%=" name="company" type="GROUP"></ACL>
     <ACL aff="neutral" cmp="%=" name="jgardner.1" type="RSV"></ACL>
     <History>
       <event state="PROCS=4" time="1302127058"></event>
     </History>
   </rsv>
 </Data>
```

Related Topics

- 6.1.4.D Resource Allocation Policy

6.1.2 Administrative Reservations

Administrative reservations behave much like standing reservations but are generally created to address non-periodic, one-time issues. All administrative reservations are created using the mrsvctl -c (or setres) command and are persistent until they expire or are removed using the mrsvctl -r (or releaseres) command.

In this topic:

6.1.2.A Annotating Administrative Reservations

Reservations can be labeled and annotated using comments allowing other admins, local users, portals and other services to obtain more detailed information regarding the reservations. Naming and annotations are configured using the `-n` and `-D` options of the *`mrsvctl`* command respectively, as in the following example:

```
> mrsvctl -c -D 'testing infiniband performance' -n nettest -h 'r:agt[15-245]'
```

6.1.2.B Using Reservation Profiles

You can set up reservation profiles to avoid manually and repetitively inputting standard reservation attributes. Profiles can specify reservation names, descriptions, ACLs, durations, hostlists, triggers, flags, and other aspects that are commonly used. With a reservation profile defined, a new administrative reservation can be created that uses this profile by specifying the `-P` flag as in the following example:

```
RSVPROFILE[mtn1] TRIGGER=AType=exec,Action="/tmp/trigger1.sh",EType=start
RSVPROFILE[mtn1] USERLIST=steve,marym
RSVPROFILE[mtn1] HOSTEXP="r:50-250"
```

```
> mrsvctl -c -P mtn1 -s 12:00:00_10/03 -d 2:00:00
```

Example 6-1: Non-Blocking System Reservations with Scheduler Pause

```
RSVPROFILE[pause] TRIGGER=atype=exec,etype=start,action="/opt/moab/bin/mschedctl -p"
RSVPROFILE[pause] TRIGGER=atype=exec,etype=cancel,action="/opt/moab/bin/mschedctl -r"
RSVPROFILE[pause] TRIGGER=atype=exec,etype=end,action="/opt/moab/bin/mschedctl -r"
```

```
> mrsvctl -c -P pause -s 12:00:00_10/03 -d 2:00:00
```

6.1.2.C Optimizing Maintenance Reservations

Any reservation causes some negative impact on cluster performance as it further limits the scheduler's ability to optimize scheduling decisions. You can mitigate this impact by using flexible ACLs and triggers.

In particular, a maintenance reservation can be configured to reduce its effective reservation shadow by allowing overlap with checkpointable/preemptible jobs until the time the reservation becomes active. This can be done using a series of triggers that perform the following actions:

- Modify the reservation to disable preemption access.
- Preempt jobs that may overlap the reservation.
- Cancel any jobs that failed to properly checkpoint and exit.

The following example highlights one possible configuration:

```
RSVPROFILE[adm1] JOBATTRLIST=PREEMPTEE
RSVPROFILE[adm1] DESCRIPTION="regular system maintenance"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-
300,AType=internal,Action="rsv:-:modify:acl:jattr-=PREEMPTEE"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-240,AType=jobpreempt,Action="checkpoint"
RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-60,AType=jobpreempt,Action="cancel"
```

```
> mrsvctl -c -P adm1 -s 12:00:00_10/03 -d 8:00:00 -h ALL
```

This reservation reserves all nodes in the cluster for a period of eight hours. Five minutes before the reservation starts, the reservation is modified to remove access to new preemptible jobs. Four minutes before the reservation starts, preemptible jobs that overlap the reservation are checkpointed. One minute before the reservation, all remaining jobs that overlap the reservation are canceled.

Related Topics

- 7.2 Backfill
- Chapter 21: Preemption
- 3.7.24 mrsvctl (command)

6.1.3 Standing Reservations

Standing reservations build upon the capabilities of advance reservations to enable you to enforce advanced usage policies in an efficient manner. Standing reservations provide a superset of the capabilities typically found in a batch queuing system's class or queue architecture. For example, queues can be used to allow only particular types of jobs access to certain compute resources. Also, some batch systems allow these queues to be configured so that they only allow this access during certain times of the day or week. Standing reservations allow these same capabilities but with greater flexibility and efficiency than is typically found in a normal queue management system.

Standing reservations provide a mechanism by which you can dedicate a particular block of resources for a special use on a regular daily or weekly basis. For example, node X could be dedicated to running jobs only from users in the accounting group every Friday from 4 to 10 P.M. See 6.1.1 Reservation Overview for more information about the use of reservations. 6.1.5 Configuring and Managing Reservations provides a detailed explanation of the concepts and steps involved in the creation and configuration of standing reservations.

A standing reservation is a powerful means of doing the following:

- Controlling local credential based access to resources.
- Controlling external peer and grid based access to resources.
- Controlling job responsiveness and turnaround.

Related Topics

- Appendix A: Moab Parameters - SRCFG parameter
- Chapter 23: Moab Workload Manager for Grids
- 3.7.15 mdiag -s (diagnose standing reservations)

6.1.4 Reservation Policies

In this topic:

6.1.4.A Controlling Priority Reservation Creation

In addition to standing and administrative reservations, Moab can also create priority reservations. These reservations are used to allow the benefits of out-of-order execution (such as is available with backfill) without the side effect of job starvation. Starvation can occur in any system where the potential exists for a job to be overlooked by the scheduler for an indefinite period. In the case of backfill, small jobs may continue to run on available resources as they become available while a large job sits in the queue, never able to find enough nodes available simultaneously on which to run.

To avoid such situations, priority reservations are created for high priority jobs that cannot run immediately. When making these reservations, the scheduler determines the earliest time the job could start and then reserves these resources for use by this job at that future time.

6.1.4.B Priority Reservation Creation Policy

Organizations have the ability to control how priority reservations are created and maintained. It is possible that one job can be at the top of the priority queue for a time and then get bypassed by another job submitted later. The parameter RESERVATIONPOLICY enables you to determine how existing reservations should be handled when new reservations are made.

Value	Description

HIGHEST	All jobs that have ever received a priority reservation up to the `RESERVATIONDEPTH` number will maintain that reservation until they run, even if other jobs later bypass them in priority value.
	For example, if there are four jobs with priorities of 8, 10, 12, and 20.
	`RESERVATIONPOLICY HIGHEST`
	`RESERVATIONDEPTH 3`
	Only jobs 20, 12, and 10 get priority reservations. Later, if a job with priority higher than 20 is submitted into the queue, it will also get a priority reservation along with the jobs listed previously. If four jobs higher than 20 were to be submitted into the queue, only three get priority reservations, in accordance with the condition set in the `RESERVATIONDEPTH` policy.
	With `HIGHEST`, Moab may appear to exceed the `RESERVATIONDEPTH` if it has already scheduled the maximum number of priority reservations and then users submit jobs with higher priority than those already given a priority reservation. Moab keeps all of the previously-created priority reservations and creates new ones for jobs with higher priority (again up to the quantity specified with `RESERVATIONDEPTH`). This means that, if your `RESERVATIONDEPTH` is set to 3, Moab can potentially schedule up to 3 new priority reservations each scheduling iteration, as long as new higher-priority jobs are continually submitted. This behavior ensures that the highest-priority jobs receive attention while the former highest-priority jobs do not lose their priority reservation.

Value	Description
CURRENTHIGHEST	Only the current top <RESERVATIONDEPTH> priority jobs receive reservations. Under this policy, all job reservations are destroyed each iteration when the queue is re-prioritized. The top jobs in the queue are then given new reservations.
NEVER	No priority reservations are made.

Priority Reservation Depth

By default, only the highest priority job receives a priority reservation. However, this behavior is configurable via the RESERVATIONDEPTH policy. Moab's default behavior of only reserving the highest priority job allows backfill to be used in a form known as liberal backfill. Liberal backfill tends to maximize system utilization and minimize overall average job turnaround time. However, it does lead to the potential of some lower priority jobs being indirectly delayed and might lead to greater variance in job turnaround time. The `RESERVATIONDEPTH` parameter can be set to a very large value, essentially enabling what is called conservative backfill where every job that cannot run is given a reservation. Most sites prefer the liberal backfill approach associated with the default `RESERVATIONDEPTH` of 1 or else select a slightly higher value. It is important to note that to prevent starvation in conjunction with reservations, monotonically increasing priority factors such as queue time or job XFactor should be enabled. See 4.1.1 Job Priority Overview for more information on priority factors.

Another important consequence of backfill and reservation depth is how they affect job priority. In Moab, all jobs are prioritized. Backfill allows jobs to be run out of order and therefore, to some extent, job priority to be ignored. This effect, known as priority dilution, can cause many site policies implemented via Moab prioritization policies to be ineffective. Setting the `RESERVATIONDEPTH` parameter to a higher value gives job priority more teeth at the cost of slightly lower system utilization. This lower utilization results from the constraints of these additional reservations, decreasing the scheduler's freedom and its ability to find additional optimizing schedules. Anecdotal evidence indicates that these utilization losses are fairly minor, rarely exceeding 8%.

It is difficult to know the right setting for the `RESERVATIONDEPTH` parameter. Surveys indicate that the vast majority of sites use the default value of 1. Sites that do modify this value typically set it somewhere in the range of 2 to 10. The following guidelines may be useful in determining if and how to adjust this parameter:

Reasons to Increase RESERVATIONDEPTH

- The estimated job start time information provided by the showstart command is heavily used and the accuracy needs to be increased.

- Priority dilution prevents certain key mission objectives from being fulfilled.
- Users are more interested in knowing when their job will run than in having it run sooner.

Reasons to Decrease RESERVATIONDEPTH

- Scheduling efficiency and job throughput need to be increased.

Assigning Per-QoS Reservation Creation Rules

QoS based reservation depths can be enabled via the RESERVATIONQOSLIST parameter. This parameter allows varying reservation depths to be associated with different sets of job QoSes. For example, the following configuration creates two reservation depth groupings:

```
RESERVATIONDEPTH[0]    8
RESERVATIONQOSLIST[0] highprio,interactive,debug
RESERVATIONDEPTH[1]    2
RESERVATIONQOSLIST[1] batch
```

This example causes that the top 8 jobs belonging to the aggregate group of `highprio`, `interactive`, and `debug` QoS jobs will receive priority reservations. Additionally, the top two `batch` QoS jobs will also receive priority reservations. Use of this feature enables sites to maintain high throughput for important jobs by guaranteeing that a significant proportion of these jobs progress toward starting through use of the priority reservation.

By default, the following parameters are set inside Moab:

```
RESERVATIONDEPTH[DEFAULT]    1
RESERVATIONQOSLIST[DEFAULT] ALL
```

This allows one job with the highest priority to get a reservation. These values can be overwritten by modifying the `DEFAULT` policy.

6.1.4.C Managing Resource Failures

Moab allows organizations to control how to best respond to a number of real-world issues. Occasionally when a reservation becomes active and a job attempts to start, various resource manager race conditions or corrupt state situations will prevent the job from starting. By default, Moab assumes the resource manager is corrupt, releases the reservation, and attempts to re-create the reservation after a short timeout. However, in the interval between the reservation release and the re-creation timeout, other priority reservations may allocate the newly available resources, reserving them before the original reservation gets an opportunity to reallocate them. Therefore, when the original job reservation is re-established, its original resource may be unavailable and the resulting new reservation might be delayed several hours from the earlier start time. The parameter RESERVATIONRETRYTIME allows a site that is experiencing frequent resource manager race conditions and/or corruption situations to tell Moab to hold on to the reserved resource for a period of time in an attempt to allow the resource manager to correct its state.

6.1.4.D Resource Allocation Policy

By default, when a standing or administrative reservation is created, Moab allocates nodes in accordance with the specified taskcount, node expression, node constraints, and the MINRESOURCE node allocation policy.

6.1.4.E Accounting for Reserved Resources

If an accounting manager is configured within Moab, resources consumed by jobs are tracked and charged by default. However, resources dedicated to a reservation are not charged by default although they are recorded within the reservation event record. In particular, total processor seconds reserved by the reservation and total processor seconds blocked by jobs are among the statistics recorded. While some of this information is available in real-time using the mdiag -r command (see the 'PH Allocated to Jobs' field), it is not written to the event log until reservation completion.

It is possible to track or charge for the total and unused cycles in a reservation with the accounting manager. See 5.5.5 Charging for Reservations for details on configuring Moab to use the accounting manager to track or charge for unused processors in a reservation.

Related Topics

- 6.1.1 Reservation Overview
- 7.2 Backfill

6.1.5 Configuring and Managing Reservations

In this topic:

6.1.5.A Reservation Attributes

All reservations possess a time frame of activity, an access control list (ACL), and a list of resources to be reserved. Additionally, reservations can also possess a number of extension attributes including epilog/prolog specification, reservation ownership and accountability attributes, and special flags that modify the reservation's behavior.

Start/End Time

All reservations possess a start and an end time that define the reservation's active time. During this active time, the resources within the reservation can only be used as specified by the reservation access control list (ACL). This active time can be specified as either a start/end pair or a start/duration pair. Reservations exist and are visible from the time they are created until the active time ends at which point they are automatically removed.

Access Control List (ACL)

For a reservation to be useful, it must be able to limit who or what can access the resources it has reserved.

By default, a reservation can allocate resources that possess credentials that meet the submitter's ACL. In other words, a user's reservation won't necessarily allocate only free and idle nodes. If a reservation exists that coincides with the submitter's ACL, the nodes under that reservation are also considered for allocation. This is referred to as ACL overlap. To make new reservations allocate *only* free and idle nodes, you must use the `NOACLOVERLAP` flag.

This is handled by way of an ACL. With reservations, ACLs can be based on credentials, resources requested, or performance metrics. In particular, with a standing reservation, the attributes `USERLIST`, `GROUPLIST`, `ACCOUNTLIST`, `CLASSLIST`, `QOSLIST`, `JOBATTRLIST`, `PROCLIMIT`, `MAXTIME`, or `TIMELIMIT` can be specified. See the sections Affinity and ACL Modifiers below.

Reservation access can be adjusted based on a job's requested node features by mapping node feature requests to job attributes as in the following example:

```
NODECFG[DEFAULT]   FEATURES=ia64
NODETOJOBATTRMAP   ia64,ia32
SRCFG[pgs]         JOBATTRLIST=ia32
```

```
> mrsvctl -c -a jattr=gpfs\! -h "r:13-500"
```

Selecting Resources

When specifying which resources to reserve, the admin has a number of options. These options allow control over how many resources are reserved and where they are reserved. The following reservation attributes allow the admin to define resources.

Task Description

Moab uses the task concept extensively for its job and reservation management. A task is simply an atomic collection of resources, such as processors, memory, or local disk, which must be found on the same node. For example, if a task requires 4 processors and 2 GB of memory, the scheduler must find all processors AND memory on the same node; it cannot allocate 3 processors and 1 GB on one node and 1 processor and 1 GB of memory on another node to satisfy this task. Tasks constrain how the scheduler must collect resources for use in a standing reservation; however, they do not constrain the way in which the scheduler makes these cumulative resources available to jobs. A job can use the resources covered by an accessible reservation in whatever way it needs. If reservation X allocates 6 tasks with 2 processors and 512 MB of memory each, it could support job Y that requires 10 tasks of 1 processor and 128 MB of memory or job Z that requires 2 tasks of 4 processors and 1 GB of memory each. The task constraints used to acquire a reservation's resources are transparent to a job requesting use of these resources.

```
SRCFG[test] RESOURCES=PROCS:2,MEM:1024
```

Taskcount

Using the task description, the taskcount attribute defines how many tasks must be allocated to satisfy the reservation request. To create a reservation, a taskcount and/or a hostlist must be specified.

```
SRCFG[test] TASKCOUNT=256
```

Hostlist

A hostlist constrains the set of resources available to a reservation. If no taskcount is specified, the reservation attempts to reserve one task on each of the listed resources. If a taskcount is specified that requests fewer resources than listed in the hostlist, the scheduler reserves only the number of tasks from the hostlist specified by the taskcount attribute. If a taskcount is specified that requests more resources than listed in the hostlist, the scheduler reserves the hostlist nodes first and then seeks additional resources outside of this list.

When specifying resources for a hostlist, you can specify *exact set*, *superset*, or *subset* of nodes on which the job must run. Use the caret (^) or asterisk (*) characters to specify a hostlist as *superset* or *subset* respectively.

- An exact set is defined without a caret or asterisk. An exact set means *all* the hosts in the specified hostlist must be selected for the job.
- A subset means the specified hostlist is used first to select hosts for the job. If the job requires more hosts than are in the subset hostlist, they will be obtained from elsewhere if possible. If the job does not require all of the nodes in the subset hostlist, it will use only the ones it needs.
- A superset means the hostlist is the *only* source of hosts that should be considered for running the job. If the job can't find the necessary resources in the superset hostlist it should *not* run. No other hosts should be considered in allocating the job.

```
SRCFG[test] HOSTLIST=node01,node1[3-5]
```

Example 6-2: Subset

```
SRCFG[one] HOSTLIST=node1,node5* TASKCOUNT=5 PERIOD=DAY USERLIST=user1
```

Example 6-3: Superset

```
SRCFG[two] HOSTLIST=node1,node2,node3,node4,node5^ TASKCOUNT=3 PERIOD=DAY
USERLIST=user1
```

Node Features

Node features can be specified to constrain which resources are considered.

Example 6-4: NODEFEATURES

```
SRCFG[test] NODEFEATURES=fastos
```

Partition

A partition can be specified to constrain which resources are considered.

Example 6-5: PARTITION

```
SRCFG[test] PARTITION=core3
```

Flags

Reservation flags allow specification of special reservation attributes or behaviors. Supported flags are listed in the following table:

Flags Table

Flag Name	Description
ACLOVERLAP	**Deprecated (this is now a default flag).** In addition to free or idle nodes, a reservation can also reserve resources with job reservations possessing credentials that meet the reservation's ACL. To change this behavior, set the NOACLOVERLAP flag.
ADVRESJOBDESTROY	All jobs that have an ADVRES matching this reservation are canceled when the reservation is destroyed.
ALLOWGRID	By default, jobs migrated from one Moab to another Moab in a grid are not allowed within local reservations. This flag allows migrated jobs to access local reservations when they match the ACL.
ALLOWJOBOVERLAP	A job is allowed to start in a reservation that may end before the job completes. When the reservation ends before the job completes, the job will not be canceled but will continue to run.
BESTEFFORT	Reservation is placed, even if only some of the specified resources are available.
BYNAME	Reservation only allows access to jobs that meet reservation ACLs and explicitly request the resources of this reservation using the job ADVRES flag. See 6.1.1.D Job to Reservation Binding.
DEDICATEDRESOURCE (a.k.a. EXCLUSIVE)	Reservation placed only on resources that are not reserved by any other reservation including job, system, and user reservation. There are two exceptions to this: • Reserved resources could be allocated when DEDICATEDRESOURCE is combined with IGNJOBRSV* • Reserved resources could be allocated when a reservation matches the submitter's ACL. In this case, to make DEDICATEDRESOURCE *truly* exclusive, use the NOACLOVERLAP flag.

Flag Name	Description
	The order that SRCFG reservations are listed in the configuration is important when using `DEDICATEDRESOURCE`, because reservations made afterwards can steal resources later. During configuration, list `DEDICATEDRESOURCE` reservations last to guarantee exclusiveness.
ENFORCENODESET	Moab will ensure global NODESET rules are followed when initially placing the reservation.
EVACVMS	Reservation will automatically evacuate virtual machines from the reservation nodelist. The same action can be accomplished by using reservation profiles. For more information, see 6.1.2.C Optimizing Maintenance Reservations.
IGNIDLEJOBS*	Reservation can be placed on top of idle job reservations. This flag is meant to be used in conjunction with `DEDICATEDRESOURCE`.
IGNJOBRSV*	Ignores existing job reservations, allowing the reservation to be forced onto available resources even if it conflicts with existing job reservations. User and system reservation conflicts are still valid. It functions the same as IGNIDLEJOBS plus allows a reservation to be placed on top of an existing running job's reservation. This flag is meant to be used in conjunction with `DEDICATEDRESOURCE`.
IGNRSV*	Request ignores existing resource reservations allowing the reservation to be forced onto available resources even if this conflicts with other reservations. It functions the same as IGNJOBRSV plus allows the reservation to be placed on top of the system reservations.

Flag Name	Description
	This flag is meant to be used in conjunction with DEDICATEDRESOURCE.
IGNSTATE*	Reservation ignores node state when assigning nodes. It functions the same as IGNRSV plus allows the reservation to be placed on nodes that are not currently available. Also ignores resource availability on nodes. IGNSTATE is specified by default when using a HOSTLIST to define nodes. However, if using a HOSTLIST and a TASKCOUNT, you need to specify IGNSTATE if you want Moab to ignore the node state when assigning nodes to the reservation.
NOACLOVERLAP	All resources must be free or idle, with no existing reservations. Moab will not allocate in-use resources even if they match the reservation's ACL. `mrsvctl -c -t 12 -E -F noacloverlap -a user==john` *Moab looks for resources that are exclusive (free). Without the flag, Moab looks for resources that are exclusive or that are already running `john`'s jobs.* This flag is meant to be used in conjunction with DEDICATEDRESOURCE.
OWNEREXCLUSIVEBF	When the owner of the reservation has an idle job in the queue only owner jobs will be allowed to backfill into the reservation. This blocks non-owner jobs from backfilling into the reservation. ENABLEPROFILING must be set for the owner credential.
OWNERPREEMPT	Jobs by the reservation owner are allowed to preempt non-owner jobs using reservation resources.

Flag Name	Description
OWNERPREEMPTIGNOREMINTIME	Allows the `OWNERPREEMPT` flag to trump the `PREEMPTMINTIME` setting for jobs already running on a reservation when the owner of the reservation submits a job. For example: without the `OWNERPREEMPTIGNOREMINTIME` flag set, a job submitted by the owner of a reservation will not preempt non-owner jobs already running on the reservation until the `PREEMPTMINTIME` setting (if set) for those jobs is passed. With the `OWNERPREEMPTIGNOREMINTIME` flag set, a job submitted by the owner of a reservation immediately preempts non-owner jobs already running on the reservation, regardless of whether `PREEMPTMINTIME` is set for the non-owner jobs.
OWNERPREEMPTQT	Specifies how much time a job from OWNER must wait in the queue before preempting jobs within the standing reservation. `SRCFG[test] OWNERPREEMPTQT=2:00:00` *OWNER jobs must wait 2 hours in the queue before preempting.*
REQFULL	Reservation is only created when all resources can be allocated.
SINGLEUSE	Reservation is automatically removed after completion of the first job to use the reserved resources.
SPACEFLEX	**Deprecated (this is now a default flag).** Reservation is allowed to adjust resources allocated over time in an attempt to optimize resource utilization.

* `IGNIDLEJOBS`, `IGNJOBRSV`, `IGNRSV`, and `IGNSTATE` flags are built on one another and form a hierarchy. `IGNJOBRSV` performs the function of `IGNIDLEJOBS` plus its own functions. `IGNRSV` performs the function of `IGNJOBSRV` and `IGNIDLEJOBS` plus its own functions. `IGNSTATE` performs the function of `IGNRSV`, `IGNJOBRSV`, and `IGNIDLEJOBS` plus its own functions. While you can use combinations of these flags, it is not necessary. If you set one flag, you do not need to set other flags that fall beneath it in the hierarchy.

Most flags can be associated with a reservation via the mrsvctl -c -F command or the SRCFG parameter.

6.1.5.B Configuring Standing Reservations

Standing reservations allow resources to be dedicated for particular uses. This dedication can be configured to be permanent or periodic, recurring at a regular time of day and/or time of week. There is extensive applicability of standing reservations for everything from daily dedicated job runs to improved use of resources on weekends. By default, standing reservations can overlap other reservations. Unless you set an ignore-type flag (`ACLOVERLAP`, `DEDICATEDRESOURCE`, `IGNIDLEJOBS`, or `IGNJOBRSV`), they are automatically given the `IGNRSV` flag. All standing reservation attributes are specified via the SRCFG parameter using the attributes listed in the table below:

Standing Reservation Attributes

ACCESS	
Format	`DEDICATED` or `SHARED`
Default	---
Description	If set to `SHARED`, allows a standing reservation to use resources already allocated to other non-job reservations. Otherwise, these other reservations block resource access.
Example	`SRCFG[test] ACCESS=SHARED` *Standing reservation `test` can access resources allocated to existing standing and administrative reservations.* The order that SRCFG reservations are listed in the configuration are important when using `DEDICATED`, because reservations made afterwards can steal resources later. During configuration, list `DEDICATED` reservations last to guarantee exclusiveness.

ACCOUNTLIST	
Format	List of valid, comma-delimited account names (see the section ACL Modifiers below).
Default	---
Description	Specifies that jobs with the associated accounts can use the resources contained within this reservation.
Example	`SRCFG[test] ACCOUNTLIST=ops,staff` *Jobs using the account `ops` or `staff` are granted access to the resources in standing reservation `test`.*

CHARGE	
Format	<BOOLEAN>
Default	---

CHARGE	
Description	Overrides the default charging behavior: • If set to `True`, overrides AMCFG[] RESERVATIONCHARGEPOLICY=Select to indicate that this reservation should be charged, even if no `ChargeAccount` or `ChargeUser` are specified. • If set to `False`, overrides AMCFG [] RESERVATIONCHARGEPOLICY=All to indicate that this reservation should not be charged. If AMCFG[] RESERVATIONCHARGEPOLICY=None, this parameter has no effect.
Example	`SRCFG[sr_maml] CHARGE=False` *Prevents charges to this reservation (e.g., when `AMCFG[] RESERVATIONCHARGEPOLICY=ALL`).*

CHARGEACCOUNT	
Format	Any valid account name.
Default	---
Description	Specifies that idle cycles for this reservation should be charged against the specified account (via the Accounting Manager). `CHARGEACCOUNT` must be used in conjunction with CHARGEUSER.

Example	`SRCFG[sr_maml] CHARGEACCOUNT=math`
	`SRCFG[sr_maml] CHARGEUSER=john`
	Moab charges all idle cycles within reservations supporting standing reservation `sr_maml` to account `math`.

CHARGEUSER	
Format	Any valid username.
Default	---

CHARGEUSER	
Description	Specifies that idle cycles for this reservation should be charged against the specified user (via the Accounting Manager). CHARGEUSER must be used in conjunction with CHARGEACCOUNT.

Example	`SRCFG[sr_mam1] CHARGEACCOUNT=math`
	`SRCFG[sr_mam1] CHARGEUSER=john`
	Moab charges all idle cycles within reservations supporting standing reservation `sr_mam1` *to user* `john`*.*

CLASSLIST	
Format	List of valid, comma-delimited classes/queues (see the section ACL Modifiers below).
Default	---
Description	Specifies that jobs with the associated classes/queues can use the resources contained within this reservation.
Example	`SRCFG[test] CLASSLIST=!interactive` *Jobs not using the class* `interactive` *are granted access to the resources in standing reservation* `test`*.*

CLUSTERLIST	
Format	List of valid, comma-delimited peer clusters (see Chapter 23: Moab Workload Manager for Grids).
Default	---
Description	Specifies that jobs originating within the listed clusters can use the resources contained within this reservation.
Example	`SRCFG[test] CLUSTERLIST=orion2,orion7` *Moab grants jobs from the listed peer clusters access to the reserved resources.*

COMMENT	
Format	<STRING> If the string contains whitespace, it should be enclosed in single (') or double quotes (").
Default	---
Description	Specifies a descriptive message associated with the standing reservation and all child reservations.
Example	`SRCFG[test] COMMENT='rsv for network testing'` *Moab annotates the standing reservation `test` and all child reservations with the specified message. These messages show up within Moab client commands, Moab web tools, and graphical administrator tools.*

DAYS	
Format	One or more of the following (comma-delimited): • `Mon` • `Tue` • `Wed` • `Thu` • `Fri` • `Sat` • `Sun` • `[ALL]`
Default	`[ALL]`
Description	Specifies which days of the week the standing reservation is active.
Example	`SRCFG[test] DAYS=Mon,Tue,Wed,Thu,Fri` *Standing reservation `test` is active Monday through Friday.*

DEPTH	
Format	<INTEGER>
Default	`2`
Description	Specifies the depth of standing reservations to be created (one per period). To satisfy the `DEPTH`, Moab creates new reservations at the beginning of the specified PERIOD. If your reservation ends at the same time that a new `PERIOD` begins, the number of reservations might not match the requested `DEPTH`. To prevent or resolve this issue, set the ENDTIME a couple minutes before the beginning of the next `PERIOD`. For example, set the `ENDTIME` to `23:58` instead of `00:00`.
Example	`SRCFG[test] PERIOD=DAY DEPTH=6` *Specifies that six reservations will be created for standing reservation `test`.*

DISABLE	
Format	<BOOLEAN>
Default	`FALSE`
Description	Specifies that the standing reservation should no longer spawn child reservations.
Example	`SRCFG[test] PERIOD=DAY DEPTH=7 DISABLE=TRUE` *Specifies that reservations are created for standing reservation `test` for today and the next six days.*

ENDTIME	
Format	[[[DD:]HH:]MM:]SS
Default	`24:00:00`
Description	Specifies the time of day the standing reservation period ends (end of day

ENDTIME	
	or end of week depending on PERIOD).

Example	`SRCFG[test] STARTTIME=8:00:00`
	`SRCFG[test] ENDTIME=17:00:00`
	`SRCFG[test] PERIOD=DAY`
	Standing reservation `test` *is active from 8:00 A.M. until 5:00 P.M.*

FLAGS	
Format	Comma-delimited list of zero or more flags listed in the Flags section above.
Default	---
Description	Specifies special reservation attributes.
Example	`SRCFG[test] FLAGS=BYNAME,DEDICATEDRESOURCE` *Jobs can only access the resources within this reservation if they explicitly request the reservation by name. Further, the reservation is created to not overlap with other reservations.*

GROUPLIST	
Format	One or more comma-delimited group names.
Default	`[ALL]`
Description	Specifies the groups allowed access to this standing reservation (see the section ACL Modifiers below).

Example	`SRCFG[test] GROUPLIST=staff,ops,special`
	`SRCFG[test] CLASSLIST=interactive`
	Moab allows jobs with the listed group IDs or which request the job class `interactive` *to use the resources covered by the standing reservation.*

HOSTLIST	
Format	One or more comma-delimited host names or *host expressions* or the string `class:<classname>`.
Default	---
Description	Specifies the set of hosts that the scheduler can search for resources to satisfy the reservation. If specified using the `class:X` format, Moab only selects hosts that support the specified class. If `TASKCOUNT` is also specified, only `TASKCOUNT` tasks are reserved. Otherwise, all matching hosts are reserved. The `HOSTLIST` attribute is treated as host regular expression so `foo10` will map to `foo10`, `foo101`, `foo1006`, and so forth. To request an exact host match, the expression can be bounded by the carat and dollar symbol expression markers as in `^foo10$`. When specifying resources for a hostlist, you can specify exact set, superset, or subset of nodes on which the job must run. Use the caret (^) or asterisk (*) characters to specify a hostlist as superset or subset respectively. See hostlist in the section Selecting Resources below for more information. When using r: ensure your node indexes are correct by customizing the NODEIDFORMAT parameter. See the parameter NODEIDFORMAT for more information.

Example	`SRCFG[test] HOSTLIST=node001,node002,node003`
	`SRCFG[test] RESOURCES=PROCS:2;MEM:512`
	`SRCFG[test] TASKCOUNT=2`
	
	Moab reserves a total of two tasks with `2` processors and `512` MB each, using resources located on `node001`, `node002`, and/or `node003`.
	
	`SRCFG[test] HOSTLIST=node01,node1[3-5]`
	
	The reservation will consume all nodes that have 'node01' somewhere in their names and all nodes that have both 'node1' and either a '3,' '4,' or '5' in their names.
	
	`SRCFG[test] HOSTLIST=r:node[1-6]`
	
	The reservation will consume all nodes with names that begin with 'node' and end with any number 1 through 6. In other words, it will reserve node1, node2, node3, node4, node5, and

HOSTLIST	
	node6.

JOBATTRLIST	
Format	Comma-delimited list of one or more of the following job attributes: • `PREEMPTEE` • `INTERACTIVE` • Any generic attribute configured through `NODECFG`
Default	---
Description	Specifies job attributes that grant a job access to the reservation. Values can be specified with a `!=`assignment to only allow jobs NOT requesting a certain feature inside the reservation. To enable/disable reservation access based on requested node features, use the parameter NODETOJOBATTRMAP. This applies to any attribute, including `PREEMPTEE` and `INTERACTIVE` (which require uppercase). `JOBATTRLIST` can serve as an ACL on its own. See the section Access Control List (ACL) above for more information.

Example	`SRCFG[test] JOBATTRLIST=PREEMPTEE`
	`NODETOJOBATTRMAP PREEMPTEE`
	Preemptible jobs can access the resources reserved within this reservation.

MAXJOB	
Format	<INTEGER>
Default	---
Description	Specifies the maximum number of jobs that can run in the reservation.

MAXJOB	
Example	`SRCFG[test] MAXJOB=1` *Only one job will be allowed to run in this reservation.*

MAXTIME	
Format	[[[DD:]HH:]MM:]SS[+]
Default	---
Description	Specifies the maximum time for jobs allowable. Can be used with Affinity to attract jobs with same `MAXTIME`. The `MAXTIME` and `TIMELIMIT` reservation attributes perform the same function, and are therefore interchangeable. Also, either will serve as an ACL, as described in Access Control List (ACL) above.
Example	`SRCFG[test] MAXTIME=1:00:00+` *Jobs with a time of `1:00:00` are attracted to this reservation.*

NODEFEATURES	
Format	Comma-delimited list of node features.
Default	---
Description	Specifies the required node features for nodes that are part of the standing reservation.
Example	`SRCFG[test] NODEFEATURES=wide,fddi` *All nodes allocated to the standing reservation must have both the `wide` and `fddi` node attributes.*

OWNER	
Format	<CREDTYPE>:<CREDID>

OWNER	
	Where `<CREDTYPE>` is one of `USER`, `GROUP`, `ACCT`, `QoS`, `CLASS` or `CLUSTER`, and `<CREDTYPE>` is a valid credential ID of that type.
Default	---
Description	Specifies the owner of the reservation. Setting ownership for a reservation grants the user management privileges, including the power to release it. Setting a `USER` as the `OWNER` of a reservation gives that user privileges to query and release the reservation. For sandbox reservations, sandboxes are applied to a specific peer only if `OWNER` is set to `CLUSTER:<PEERNAME>`.
Example	`SRCFG[test] OWNER=ACCT:jupiter` *User `jupiter` owns the reservation and can be granted special privileges associated with that ownership.*

PARTITION	
Format	Valid partition name.
Default	`[ALL]`
Description	Specifies the partition in which to create the standing reservation.
Example	`SRCFG[test] PARTITION=OLD` *The standing reservation will only select resources from partition OLD.*

PERIOD	
Format	One of `DAY`, `WEEK`, or `INFINITY`
Default	`DAY`
Description	Specifies the `period` of the standing reservation.

PERIOD	
Example	`SRCFG[test] PERIOD=WEEK` Each standing reservation covers a one week period.

PROCLIMIT	
Format	<QUALIFIER><INTEGER> `<QUALIFIER>` can be one of the following <, <=, ==, >=, >
Default	---
Description	Specifies the processor limit for jobs requesting access to this standing reservation. `PROCLIMIT` can serve as an ACL on its own. See the section Access Control List (ACL) above for more information.
Example	`SRCFG[test] PROCLIMIT<=4` *Jobs requesting* `4` *or fewer processors are allowed to run.*

PSLIMIT	
Format	<QUALIFIER><INTEGER> `<QUALIFIER>` can be one of the following <, <=, ==, >=, >
Default	---
Description	Specifies the processor-second limit for jobs requesting access to this standing reservation.
Example	`SRCFG[test] PSLIMIT<=40000` *Jobs requesting* `40000` *or fewer processor-seconds are allowed to run.*

QOSLIST	
Format	Zero or more valid, comma-delimited QoS names.
Default	---
Description	Specifies that jobs with the listed QoS names can access the reserved resources.
Example	`SRCFG[test] QOSLIST=hi,low,special` *Moab allows jobs using the listed QOSs access to the reserved resources.*

REQUIREDACCTLIST	
Format	One or more comma-delimited accounts.
Default	---
Description	When present, any jobs in the reservation must match one of the listed accounts. This attribute can also be used in conjunction with REQUIREDUSERLIST. If both REQUIREDACCTLIST and REQUIREDUSERLIST are specified, all jobs in the reservation must match both. We recommend that any entries in the REQUIREDACCTLIST be present in the ACCOUNTLIST attribute to handle reservation affinities.
Example	`SRCFG[test] REQUIREDUSERLIST=john,bob USERLIST=john,bob` `SRCFG[test] REQUIREDACCTLIST=eng,chem ACCOUNTLIST=eng,chem` *A job must belong to either user 'john' or 'bob' AND either account 'eng' or 'chem'.*

REQUIREDTPN	
Format	<QUALIFIER><INTEGER> `<QUALIFIER>` can be one of the following <, <=, ==, >=, >

REQUIREDTPN	
Default	---
Description	Restricts access to reservations based on the job's TPN (tasks per node).
Example	`SRCFG[test] REQUIREDTPN==4` *Jobs with tpn=4 or ppn=4 are allowed within the reservation, but any other TPN value are not. For more information, see the attribute TPN (Exact Tasks Per Node) below.*

REQUIREDUSERLIST	
Format	One or more comma-delimited accounts.
Default	---
Description	When present, any jobs in the reservation must match one of the listed users. This attribute can also be used in conjunction with REQUIREDACCTLIST. If both REQUIREDACCTLIST and REQUIREDUSERLIST are specified, all jobs in the reservation must match both. We recommend that any entries in the REQUIREDUSERLIST be present in the USERLIST attribute to handle reservation affinities.

Example	`SRCFG[test] REQUIREDUSERLIST=john,bob USERLIST=john,bob`
	`SRCFG[test] REQUIREDACCTLIST=eng,chem ACCOUNTLIST=eng,chem`
	A job must belong to either user 'john' or 'bob' AND either account 'eng' or 'chem'.

RESOURCES	
Format	Semicolon-delimited <ATTR>:<VALUE> pairs, where `<ATTR>` can be one of `PROCS`, `MEM`, `SWAP`, `DISK`, or `GRES`.
Default	`PROCS:-1` (all processors available on node)
Description	Specifies what resources constitute a single standing reservation task. (Each task must be able to obtain all of its resources as an atomic unit on a single

RESOURCES	
	node.) Supported resources currently include the following: • `PROCS` (number of processors) • `MEM` (real memory in MB) • `SWAP` (virtual memory in MB) • `DISK` (local disk in MB) • `GRES` (generic resource specified in the format GRES:<GRESNAME>[:<COUNT>])
Example	`SRCFG[test] RESOURCES=PROCS:1;MEM:512;GRES=matlab:3;GRES=fluent:12` *Each standing reservation task reserves 1 processor, 512 MB of real memory, 3 matlab generic resources and 12 fluent generic resources.*

ROLLBACKOFFSET	
Format	[[[DD:]HH:]MM:]SS
Default	---
Description	Specifies the minimum time in the future at which the reservation can start. This offset is rollback meaning the start time of the reservation will continuously roll back into the future to maintain this offset. Rollback offsets are a good way of providing guaranteed resource access to users under the conditions that they must commit their resources in the future or lose dedicated access. See 6.3 Quality of Service (QoS) Facilities for more information about quality of service and service level agreements; also see the section Rollback Reservations below. Neither credlock nor advres is compatible on the jobs submitted for this reservation.

Example	`SRCFG[ajax] ROLLBACKOFFSET=24:00:00 TASKCOUNT=32`
	`SRCFG[ajax] PERIOD=INFINITY ACCOUNTLIST=ajax`
	The standing reservation guarantees access to up to 32 processors within 24 hours to jobs from the ajax account.
	Adding an asterisk to the `ROLLBACKOFFSET` value pins rollback reservation start times when an idle reservation is created in the rollback reservation. For example:

ROLLBACKOFFSET	
	`SRCFG[staff] ROLLBACKOFFSET=18:00:00* PERIOD=INFINITY`

RSVACCESSLIST	
Format	<RESERVATION>[,...]
Default	---
Description	A list of reservations to which the specified reservation has access.
Example	`SRCFG[test] RSVACCESSLIST=rsv1,rsv2,rsv3`

RSVGROUP	
Format	<STRING>
Default	---
Description	See 6.1.1.G Reservation Group for a detailed description.

Example	`SRCFG[test] RSVGROUP=rsvgrp1`
	`SRCFG[ajax] RSVGROUP=rsvgrp1`

STARTTIME	
Format	[[[DD:]HH:]MM:]SS
Default	`00:00:00:00` (midnight)
Description	Specifies the time of day/week the standing reservation becomes active. Whether this indicates a time of day or time of week depends on the setting of the PERIOD attribute. If specified within a reservation profile, a value of 0 indicates the reservation should start at the earliest opportunity.

STARTTIME	

Example	`SRCFG[test] STARTTIME=08:00:00`
	`SRCFG[test] ENDTIME=17:00:00`
	`SRCFG[test] PERIOD=DAY`
	The standing reservation will be active from 8:00 A.M. until 5:00 P.M. each day.

TASKCOUNT	
Format	<INTEGER>
Default	`0` (unlimited tasks)
Description	Specifies how many tasks should be reserved for the reservation.

Example	`SRCFG[test] RESOURCES=PROCS:1;MEM:256`
	`SRCFG[test] TASKCOUNT=16`
	Standing reservation `test` reserves `16` tasks worth of resources; in this case, 16 processors and 4 GB of real memory.

TIMELIMIT	
Format	[[[DD:]HH:]MM:]SS
Default	`-1` (no time based access)
Description	Specifies the maximum allowed overlap between the standing reservation and a job requesting resource access. The MAXTIME and TIMELIMIT reservation attributes perform the same function, and are therefore interchangeable. Also, either will serve as an ACL, as described in Access Control List (ACL), above.
Example	`SRCFG[test] TIMELIMIT=1:00:00` *Moab allows jobs to access up to one hour of resources in the standing reservation.*

TPN (Exact Tasks Per Node)	
Format	<INTEGER>
Default	0 (no TPN constraint)
Description	Specifies the exact number of tasks per node that must be available on eligible nodes.

Example	`SRCFG[2] TPN=4`
	 `SRCFG[2] RESOURCES=PROCS:2;MEM:256`
	 Moab must locate four tasks on each node that is to be part of the reservation. That is, each node included in standing reservation 2 must have 8 processors and 1 GB of memory available.

TRIGGER	
Format	See 17.2.1 Creating a Trigger for syntax.
Default	N/A
Descriptio n	Specifies event triggers to be launched by the scheduler under the scheduler's ID. These triggers can be used to conditionally cancel reservations, or launch various actions at specified event offsets. See 17.1 About Object Triggers for more detail.

Example	`SRCFG[fast]`
	 `TRIGGER=EType=start,Offset=5:00:00,AType=exec,Action="/usr/local/domail.pl"`
	 Moab launches the `domail.pl` script 5 hours after any `fast` reservation starts.

USERLIST	
Format	Comma-delimited list of users.
Default	---
Description	Specifies which users have access to the resources reserved by this reservation (see the section ACL Modifiers below).

USERLIST	
Example	`SRCFG[test] USERLIST=bob,joe,mary` *Users `bob`, `joe` and `mary` can all access the resources reserved within this reservation.*

Standing Reservation Overview

A standing reservation is similar to a normal administrative reservation in that it also places an access control list on a specified set of resources. Resources are specified on a per-task basis and currently include processors, local disk, real memory, and swap. The access control list supported for standing reservations includes users, groups, accounts, job classes, and QoS levels. Standing reservations can be configured to be permanent or periodic on a daily or weekly basis and can accept a daily or weekly start and end time. Regardless of whether permanent or recurring on a daily or weekly basis, standing reservations are enforced using a series of reservations, extending a number of periods into the future as controlled by the `DEPTH` attribute of the SRCFG parameter.

The following examples demonstrate possible configurations specified with the `SRCFG` parameter.

Basic Business Hour Standing Reservation

```
SRCFG[interactive] TASKCOUNT=6 RESOURCES=PROCS:1,MEM:512
SRCFG[interactive] PERIOD=DAY DAYS=MON,TUE,WED,THU,FRI
SRCFG[interactive] STARTTIME=9:00:00 ENDTIME=17:00:00
SRCFG[interactive] CLASSLIST=interactive
```

When using the `SRCFG` parameter, attribute lists must be delimited using the comma (,), pipe (|), or colon (:) characters; they cannot be space delimited. For example, to specify a multi-class ACL, specify:

```
SRCFG[test] CLASSLIST=classA,classB
```

Only one STARTTIME and one ENDTIME value can be specified per reservation. If varied start and end times are desired throughout the week, complementary standing reservations should be created. For example, to establish a reservation from 8:00 P.M. until 6:00 A.M. the next day during business days, two reservations should be created-one from 8:00 P.M. until midnight, and the other from midnight until 6:00 A.M. Jobs can run across reservation boundaries allowing these two reservations to function as a single reservation that spans the night. The following example demonstrates how to span a reservation across 2 days on the same nodes:

```
SRCFG[Sun] PERIOD=WEEK
SRCFG[Sun] STARTTIME=00:20:00:00 ENDTIME=01:00:00:00
SRCFG[Sun] HOSTLIST=node01,node02,node03

SRCFG[Mon] PERIOD=WEEK
SRCFG[Mon] STARTTIME=01:00:00:00 ENDTIME=01:06:00:00
SRCFG[Sun] HOSTLIST=node01,node02,node03
```

The preceding example fully specifies a reservation including the quantity of resources requested using the TASKCOUNT and RESOURCES attributes. In all cases, resources are allocated to a reservation in units called tasks where a task is a collection of resources that must be allocated together on a single node. The TASKCOUNT attribute specifies the number of these tasks that should be reserved by the reservation. In conjunction with this attribute, the RESOURCES attribute defines the reservation task by indicating what resources must be included in each task. In this case, the scheduler must locate and reserve 1 processor and 512 MB of memory together on the same node for each task requested.

As mentioned previously, a standing reservation reserves resources over a given time frame. The PERIOD attribute can be set to a value of DAY, WEEK, or INFINITY to indicate the period over which this reservation should recur. If not specified, a standing reservation recurs on a daily basis. If a standing reservation is configured to recur daily, the attribute DAYS can be specified to indicate which days of the week the reservation should exist. This attribute takes a comma-delimited list of days where each day is specified as the first three letters of the day in all capital letters: MON or FRI. The preceding example specifies that this reservation is periodic on a daily basis and should only exist on business days.

The time of day during which the requested tasks are to be reserved is specified using the STARTTIME and ENDTIME attributes. These attributes are specified in standard military time HH:MM:SS format and both STARTTIME and ENDTIME specification is optional defaulting to midnight at the beginning and end of the day respectively. In the preceding example, resources are reserved from 9:00 A.M. until 5:00 P.M. on business days.

The final aspect of any reservation is the access control list indicating who or what can use the reserved resources. In the preceding example, the CLASSLIST attribute is used to indicate that jobs requesting the class 'interactive' should be allowed to use this reservation.

Specifying Reservation Resources

In most cases, only a small subset of standing reservation attributes must be specified in any given case. For example, by default, RESOURCES is set to PROCS=-1, which indicates that each task should reserve all of the processors on the node on which it is located. This, in essence, creates a one task equals one node mapping. In many cases, particularly on uniprocessor systems, this default behavior might be easiest to work with. However, in SMP environments, the RESOURCES attribute provides a powerful means of specifying an exact, multi-dimensional resource set.

The default value of PERIOD is DAYS, therefore, specifying this parameter in the preceding above was unnecessary. It was used only to introduce this parameter and indicate that other options exist beyond daily standing reservations.

Host Constrained Standing Reservation

Although the first example did specify a quantity of resources to reserve, it did not specify where the needed tasks were to be located. If this information is not specified, Moab attempts to locate the needed resources anywhere it can find them. The Example 1 reservation essentially discovers hosts where the needed resources can be found. If the SPACEFLEX reservation flag is set, then the reservation continues to float to the best hosts over the life of the reservation. Otherwise, it will be locked to the initial set of allocated hosts.

If you wanted to constrain a reservation to a subset of available resources, this could be accomplished using the HOSTLIST attribute. The HOSTLIST attribute is specified as a comma-separated list of hostnames and constrains the scheduler to only select tasks from the specified list. This attribute can exactly specify hosts or specify them using host regular expressions. The following example demonstrates a possible use of the HOSTLIST attribute:

```
SRCFG[interactive] DAYS=MON,TUE,WED,THU,FRI
SRCFG[interactive] PERIOD=DAY
SRCFG[interactive] STARTTIME=10:00:00 ENDTIME=15:00:00
SRCFG[interactive] RESOURCES=PROCS:2,MEM:256
SRCFG[interactive] HOSTLIST=node001,node002,node005,node020
SRCFG[interactive] TASKCOUNT=6
SRCFG[interactive] CLASSLIST=interactive
```

Note that the HOSTLIST attribute specifies a non-contiguous list of hosts. Any combination of hosts can be specified and hosts can be specified in any order. In this example, the TASKCOUNT attribute is also specified. These two attributes both apply constraints on the scheduler with HOSTLIST specifying where the tasks can be located and TASKCOUNT indicating how many total tasks can be allocated. In this example, six tasks are requested but only four hosts are specified. To handle this, if adequate resources are available, the scheduler may attempt to allocate more than one task per host. For example, assume that each host is a quad-processor system with 1 GB of memory. In such a case, the scheduler could allocate up to two tasks per host and even satisfy the TASKCOUNT constraint without using all of the hosts in the hostlist.

It is important to note that even if there is a one to one mapping between the value of TASKCOUNT and the number of hosts in HOSTLIST, the scheduler will not necessarily place one task on each host. If, for example, node001 and node002 were 8 processor SMP hosts with 1 GB of memory, the scheduler could locate up to four tasks on each of these hosts fully satisfying the reservation taskcount without even partially using the remaining hosts. (Moab will place tasks on hosts according to the policy specified with the NODEALLOCATIONPOLICY parameter.) If the hostlist provides more resources than what is required by the reservation as specified via TASKCOUNT, the scheduler will simply select the needed resources within the set of hosts listed.

Enforcing Policies Via Multiple Reservations

Single reservations enable multiple capabilities. Combinations of reservations can further extend a site's capabilities to impose specific policies.

Reservation Stacking

If HOSTLIST is specified but TASKCOUNT is not, the scheduler will pack as many tasks as possible onto all of the listed hosts. For example, assume the site added a second standing reservation named *debug* to its configuration that reserved resources for use by certain members of its staff using the following configuration:

```
SRCFG[interactive] DAYS=MON,TUE,WED,THU,FRI
SRCFG[interactive] PERIOD=DAY
SRCFG[interactive] STARTTIME=10:00:00 ENDTIME=15:00:00
SRCFG[interactive] RESOURCES=PROCS:2,MEM:256
SRCFG[interactive] HOSTLIST=node001,node002,node005,node020
SRCFG[interactive] TASKCOUNT=6
SRCFG[interactive] CLASSLIST=interactive
SRCFG[debug]       HOSTLIST=node001,node002,node003,node004
SRCFG[debug]       USERLIST=helpdesk
SRCFG[debug]       GROUPLIST=operations,sysadmin
SRCFG[debug]       PERIOD=INFINITY
```

The new standing reservation is quite simple. Since RESOURCES is not specified, it will allocate all processors on each host that is allocated. Since TASKCOUNT is not specified, it will allocate every host listed in HOSTLIST. Since PERIOD is set to INFINITY, the reservation is always in force and there is no need to specify STARTTIME, ENDTIME, or DAYS.

The standing reservation has two access parameters set using the attributes USERLIST and GROUPLIST. This configuration indicates that the reservation can be accessed if any one of the access lists specified is satisfied by the job. In essence, reservation access is logically ORed allowing access if the requester meets any of the access constraints specified. In this example, jobs submitted by either user `helpdesk` or any member of the group's `operations` or `sysadmin` can use the reserved resources (see the section ACL Modifiers below).

Unless ACL Modifiers are specified, access is granted to the logical *OR* of access lists specified within a standing reservation and granted to the logical *AND* of access lists across different standing reservations. A comparison of the standing reservations `interactive` and `debug` in the preceding example indicates that they both can allocate hosts `node001` and `node002`. If `node001` had both of these reservations in place simultaneously and a job attempted to access this host during business hours when standing reservation `interactive` was active. The job could only use the *doubly* reserved resources if it requests the run class `interactive` and it meets the constraints of reservation `debug`—that is, that it is submitted by user `helpdesk` or by a member of the group `operations` or `sysadmin`.

As a rule, the scheduler does not stack reservations unless it must. If adequate resources exist, it can allocate reserved resources side by side in a single SMP host rather than on top of each other. In the case of a 16 processor SMP host with two 8 processor standing reservations, 8 of the processors on this host will be allocated to the first reservation, and 8 to the next. Any configuration is possible. The 16 processor hosts can also have 4 processors reserved for user 'John,' 10 processors reserved for group 'Staff,' with the remaining 2 processors available for use by any job.

Stacking reservations is not usually required but some site admins choose to do it to enforce elaborate policies. There is no problem with doing so as long as you can keep things straight. It really is not too difficult a concept; it just takes a little getting used to. See 6.1.1 Reservation Overview for a more detailed description of reservation use and constraints.

As mentioned earlier, by default the scheduler enforces standing reservations by creating a number of reservations where the number created is controlled by the `DEPTH` attribute. Each night at midnight, the scheduler updates its periodic non-floating standing reservations. By default, `DEPTH` is set to 2, meaning when the scheduler starts up, it will create two 24-hour reservations covering a total of two days' worth of time-a reservation for today and one for tomorrow. For daily reservations, at midnight, the reservations roll, meaning today's reservation expires and is removed, tomorrow's reservation becomes today's, and the scheduler creates a new reservation for the next day.

With this model, the scheduler continues creating new reservations in the future as time moves forward. Each day, the needed resources are always reserved. At first, all appears automatic but the standing reservation `DEPTH` attribute is in fact an important aspect of reservation rollback, which helps address certain site specific environmental factors. This attribute remedies a situation that might occur when a job is submitted and cannot run immediately because the system is backlogged with jobs. In such a case, available resources might not exist for several days out and the scheduler must reserve these future resources for this job. With the default DEPTH setting of two, when midnight arrives, the scheduler attempts to roll its standing reservations but a problem arises in that the job has now allocated the resources needed for the standing reservation two days out. Moab cannot reserve the resources for the standing reservation because they are already claimed by the job. The standing reservation reserves what it can but because all needed resources are not available, the resulting reservation is now smaller than it should be, or is possibly even empty.

If a standing reservation is smaller than it should be, the scheduler will attempt to add resources each iteration until it is fully populated. However, in the case of this job, the job is not going to release its reserved resources until it completes and the standing reservation cannot claim them until this time. The DEPTH attribute enables you to specify how deep into the future a standing reservation should reserve its resources allowing it to claim the resources first and prevent this problem. If a partial standing reservation is detected on a system, it might be an indication that the reservation's DEPTH attribute should be increased.

In Example 3, the PERIOD attribute is set to INFINITY. With this setting, a single, permanent standing reservation is created and the issues of resource contention do not exist. While this eliminates the contention issue, infinite length standing reservations cannot be made periodic.

Multiple ACL Types

In most cases, access lists within a reservation are logically ORed together to determine reservation access. However, exceptions to this rule can be specified by using the required ACL marker-the asterisk (*). Any ACL marked with this symbol is required and a job is only allowed to use a reservation if it meets all required ACLs and at least one non-required ACL (if specified). A common use for this facility is in conjunction with the TIMELIMIT attribute. This attribute controls the length of time a job can use the resources within a standing reservation. This access mechanism can be ANDed or ORed to the cumulative set of all other access lists as specified by the required ACL marker. Consider the following example configuration:

```
SRCFG[special] TASKCOUNT=32
SRCFG[special] PERIOD=WEEK
SRCFG[special] STARTTIME=1:08:00:00
SRCFG[special] ENDTIME=5:17:00:00
SRCFG[special] NODEFEATURES=largememory
SRCFG[special] TIMELIMIT=1:00:00*
SRCFG[special] QOSLIST=high,low,special-
SRCFG[special] ACCOUNTLIST=!projectX,!projectY
```

The above configuration requests 32 tasks, which translate to 32 nodes. The PERIOD attribute makes this reservation periodic on a weekly basis while the attributes STARTTIME and ENDTIME specify the week offsets when this reservation is to start and end (note that the specification format has changed to DD:HH:MM:SS). In this case, the reservation starts on Monday at 8:00 A.M. and runs until Friday at 5:00 P.M. The reservation is enforced as a series of weekly reservations that only cover the specified time frame. The `NODEFEATURES` attribute indicates that each of the reserved nodes must have the node feature 'largememory' configured.

As described earlier, `TIMELIMIT` indicates that jobs using this reservation can only use it for one hour. This means the job and the reservation can only overlap for one hour. Clearly jobs requiring an hour or less of wallclock time meet this constraint. However, a four-hour job that starts on Monday at 5:00 A.M. or a 12-hour job that starts on Friday at 4:00 P.M. also satisfies this constraint. Also, note the `TIMELIMIT` required ACL marker, *; it is set indicating that jobs must not only meet the `TIMELIMIT` access constraint but must also meet one or more of the other access constraints. In this example, the job can use this reservation if it can use the access specified via `QOSLIST` or `ACCOUNTLIST`; that is, it is assigned a QoS of `high`, `low`, or `special` , or the submitter of the job has an account that satisfies the `!projectX` and `!projectY` criteria. See 6.3 Quality of Service (QoS) Facilities for more information about QoS configuration and usage.

Affinity

Reservation ACLs allow or deny access to reserved resources but they can be configured to also impact a job's affinity for a particular reservation. By default, jobs gravitate toward reservations through a mechanism known as positive affinity. This mechanism allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs that might not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use only as a last resort-using the reserved resources only when there are no other resources available. This last resort behavior is known as negative affinity. Note the '-' (hyphen or negative sign) following the `special` in the `QOSLIST` values. This special mark indicates that QoS `special` should be granted access to this reservation but should be assigned negative affinity. Therefore, the `QOSLIST` attribute specifies that QoS `high` and `low` should be granted access with positive affinity (use the reservation first where possible) and QoS `special` granted access with negative affinity (use the reservation only when no other resources are available).

Affinity status is granted on a per access object basis rather than a per access list basis and always defaults to positive affinity. In addition to negative affinity, neutral affinity can also be specified using the equal sign (=) as in `QOSLIST[0] normal= high debug= low-`.

When a job matches multiple ACLs for a reservation, the final node affinity for the node, job, and reservation combination is based on the last matching ACL entry found in the configuration file.

For example, given the following reservation ACLs, a job matching both will receive a negative affinity:

```
SRCFG[res1] USERLIST=joe+ MAXTIME<=4:00:00-
```

With the following reservation ACLs, a job matching both will receive a positive affinity:

```
SRCFG[res1] MAXTIME<=4:00:00- USERLIST=joe+
```

To configure the behavior when multiple reservations with varying affinities are on the same node, see the parameter NODEAFFINITYPOLICY.

ACL Modifiers

ACL modifiers allow you to change the default behavior of ACL processing. By default, a reservation can be accessed if one or more of its ACLs can be met by the requestor. This behavior can be changed using the following modifiers:

Not	
Symbol:	! (exclamation point)
Description	If attribute is met, the requestor is denied access regardless of any other satisfied ACLs.
Example	`SRCFG[test] GROUPLIST=staff USERLIST=!steve` *Allow access to all staff members other than `steve`.*

Required	
Symbol:	* (asterisk)
Description	All required ACLs must be satisfied for requestor access to be granted.
Example	`SRCFG[test] QOSLIST=*high MAXTIME=*2:00:00` *Only jobs in QoS `high` that request less than 2 hours of walltime are granted access.*

XOR	
Symbol:	^ (carat)
Description	All attributes of the type specified other than the ones listed in the ACL satisfy the ACL.

XOR	
Example	`SRCFG[test] QOSLIST=^high` *All jobs other than those requesting QoS `high` are granted access.*

CredLock	
Symbol:	& (ampersand)
Description	Matching jobs will be required to run on the resources reserved by this reservation. You can use this modifier on accounts, classes, groups, qualities of service, and users.
Example	`SRCFG[test] USERLIST=&john` *All of user `john`'s jobs must run in this reservation.*

HPEnable (hard policy enable)	
Symbol:	~ (tilde)
Description	ACLs marked with this modifier are ignored during soft policy scheduling and are only considered for hard policy scheduling once all eligible soft policy jobs start.
Example	`SRCFG[johnspace] USERLIST=john CLASSLIST=~debug` *All of user `john`'s jobs are allowed to run in the reservation at any time. `Debug` jobs are also allowed to run in this reservation but are only considered after all of John's jobs are given an opportunity to start. User `john`'s jobs are considered before debug jobs regardless of job priority.* If HPEnable and Not markers are used in conjunction, then specified credentials are *blocked-out* of the reservation during soft-policy scheduling.

Note the `ACCOUNTLIST` values in are preceded with an exclamation point, or NOT symbol. This indicates that all jobs with accounts other than `projectX` and `projectY` meet the account ACL. Note that if a !<X> value (!projectX) appears in an ACL line, that ACL is satisfied by any object not explicitly listed by a NOT entry. Also, if an object matches a NOT entry, the associated job is excluded from the reservation even if it meets other ACL requirements. For example, a QoS 3 job requesting account `projectX` is denied access to the reservation even though the job QoS matches the QoS ACL.

Example 6-6: Binding Users to Reservations at Reservation Creation

```
# create a 4 node reservation for john and bind all of john's jobs to that reservation
> mrsvctl -c -a user=&john -t 4
```

Reservation Ownership

Reservation ownership enables you to control who owns the reserved resources during the reservation time frame. Depending on needs, this ownership can be identical to, a subset of, or completely distinct from the reservation ACL. By default, reservation ownership implies resource accountability and resources not consumed by jobs are accounted against the reservation owner. In addition, ownership can also be associated with special privileges within the reservation.

Ownership is specified using the `OWNER` attribute in the format `<CREDTYPE>:<CREDID>`, as in `OWNER=USER:john`. To enable `john`'s jobs to preempt other jobs using resources within the reservation, the `SRCFG` attribute `FLAG` should be set to OWNERPREEMPT. In the example below, the `jupiter` project chooses to share resources with the `saturn` project but only when it does not currently need them.

Limited Shared Access

```
ACCOUNTCFG[jupiter] PRIORITY=10000
SRCFG[jupiter] HOSTLIST=node0[1-9]
SRCFG[jupiter] PERIOD=INFINITY
SRCFG[jupiter] ACCOUNTLIST=jupiter,saturn-
SRCFG[jupiter] OWNER=ACCT:jupiter
SRCFG[jupiter] FLAGS=OWNERPREEMPT
```

Partitions

A reservation can be used in conjunction with a partition. Configuring a standing reservation on a partition allows constraints to be (indirectly) applied to a partition.

Example 6-7: Time Constraints by Partition

The following example places a 3-day wall-clock limit on two partitions and a 64 processor-hour limit on jobs running on partition `small`:

```
SRCFG[smallrsv] PARTITION=small MAXTIME=3:00:00:00 PSLIMIT<=230400 HOSTLIST=ALL
SRCFG[bigrsv] PARTITION=big MAXTIME=3:00:00:00 HOSTLIST=ALL
```

Resource Allocation Behavior

As mentioned, standing reservations can operate in one of two modes, floating, or non-floating (essentially node-locked). A floating reservation is created when the flag `SPACEFLEX` is specified. If a reservation is non-floating, the scheduler allocates all resources specified by the HOSTLIST parameter regardless of node state, job load, or even the presence of other standing reservations. Moab interprets the request for a non-floating reservation as, "I want a reservation on these exact nodes, no matter what!"

If a reservation is configured to be floating, the scheduler takes a more relaxed stand, searching through all possible nodes to find resources meeting standing reservation constraints. Only Idle, Running, or Busy nodes are considered and further, only considered if no reservation conflict is detected. The reservation attribute ACCESS modifies this behavior slightly and allows the reservation to allocate resources even if reservation conflicts exist.

If a TASKCOUNT is specified with or without a HOSTEXPRESSION, Moab will, by default, only consider 'up' nodes for allocation. To change this behavior, the reservation flag IGNSTATE can be specified as in the following example:

```
SRCFG[nettest] GROUPLIST=sysadm
SRCFG[nettest] FLAGS=IGNSTATE
SRCFG[nettest] HOSTLIST=node1[3-8]
SRCFG[nettest] STARTTIME=9:00:00
SRCFG[nettest] ENDTIME=17:00:00
```

Access to existing reservations can be controlled using the reservation flag IGNRSV.

Other standing reservation attributes not covered here include PARTITION and CHARGEACCOUNT. These parameters are described in some detail in Appendix A: Moab Parameters.

Using Reservations to Guarantee Turnover

In some cases, it is desirable to make certain a portion of a cluster's resources are available within a specific time frame. The following example creates a floating reservation belonging to the jupiter account that guarantees 16 tasks for use by jobs requesting up to one hour:

```
SRCFG[shortpool] OWNER=ACCT:jupiter
SRCFG[shortpool] FLAGS=SPACEFLEX
SRCFG[shortpool] MAXTIME=1:00:00
SRCFG[shortpool] TASKCOUNT=16
SRCFG[shortpool] STARTTIME=9:00:00
SRCFG[shortpool] ENDTIME=17:00:00
SRCFG[shortpool] DAYS=Mon,Tue,Wed,Thu,Fri
```

This reservation enables a capability similar to what was known in early Maui releases as 'shortpool'. The reservation covers every weekday from 9:00 A.M. to 5:00 P.M., reserving 16 tasks and allowing jobs to overlap the reservation for up to one hour. The SPACEFLEX flag indicates that the reservation can be dynamically modified--over time to re-locate to more optimal resources. In the case of a reservation with the MAXTIME ACL, this would include migrating to resources that are in use but that free up within the MAXTIME time frame. Additionally, because the `MAXTIME` ACL defaults to positive affinity, any jobs that fit the ACL attempt to use available reserved resources first before looking elsewhere.

Rollback Reservations

Rollback reservations are enabled using the ROLLBACKOFFSET attribute and can be used to allow users guaranteed access to resources, but the guaranteed access is limited to a time-window in the future. This functionality forces users to commit their resources in the future or lose access. In Iteration 1 of the diagram below, a rollback reservation is in place for nodes 2 and 3 for a 6.5 hour block, 2 hours in the future. If the user for whom the reservation is made does not make use of the reserved nodes, other jobs are scheduled on the nodes. In Iteration 2, the rollback reservation remains in place for the same nodes, 2 hours in the future.

Image 6-2: Rollback reservation Iteration 1

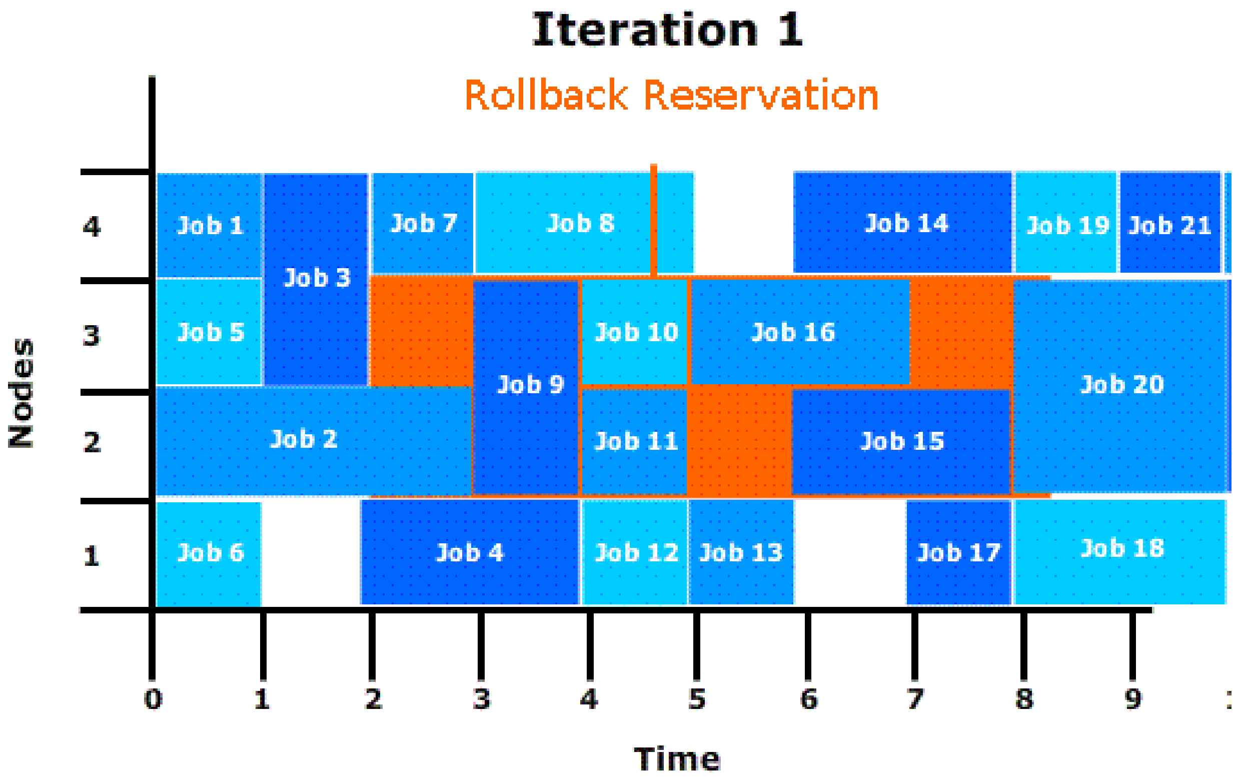
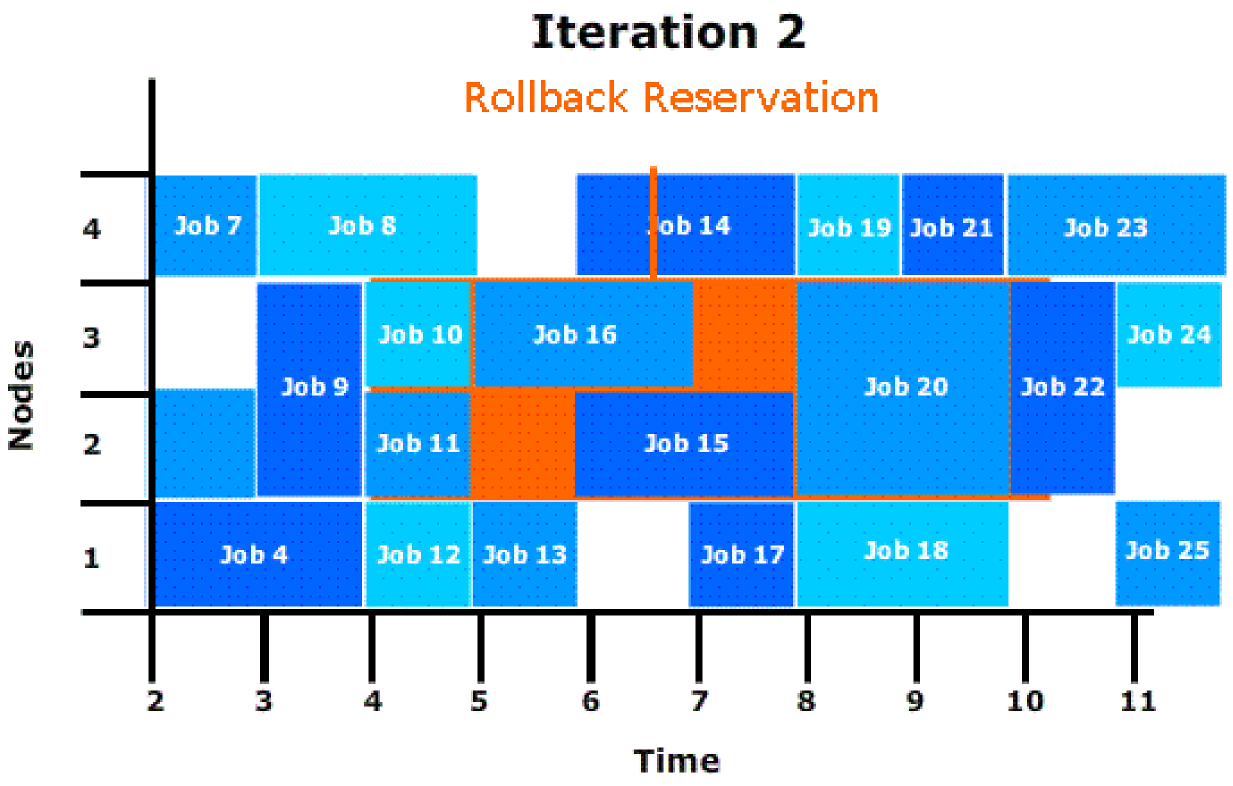


Image 6-3: Rollback reservation Iteration 2



Rollback Reservations

```
SRCFG[ajax] ROLLBACKOFFSET=24:00:00 TASKCOUNT=32
SRCFG[ajax] PERIOD=INFINITY ACCOUNTLIST=ajax
```

Adding an asterisk to the `ROLLBACKOFFSET` value pins rollback reservation start times when an idle reservation is created in the rollback reservation. For example: `SRCFG [staff] ROLLBACKOFFSET=18:00:00* PERIOD=INFINITY`.

6.1.5.C Managing Administrative Reservations

A default reservation with no ACL is termed an *administrative* reservation, but is occasionally referred to as a *system* reservation. It blocks access to all jobs because it possesses an empty access control list. It is often useful when performing administrative tasks but cannot be used for enforcing resource usage policies.

Administrative reservations are created and managed using the mrsvctl command. With this command, all aspects of reservation time frame, resource selection, and access control can be dynamically modified. The mdiag -r command can be used to view configuration, state, allocated resource information, and identify any potential problems with the reservation. The following table briefly summarizes commands used for common actions. More detailed information is available in the command summaries.

Action	Command
create reservation	mrsvctl -c <RSV_DESCRIPTION>
list reservations	mrsvctl -l
release reservation	mrsvctl -r <RSVID>
modify reservation	mrsvctl -m <ATTR>=<VAL> <RSVID>
query reservation configuration	mdiag -r <RSVID>
display reservation hostlist	mrsvctl -q resources <RSVID>

Related Topics

- Appendix A: Moab Parameters - SRCFG parameter (configure standing reservations)
- Appendix A: Moab Parameters - RSVPROFILE (create reservation profiles)

6.1.6 Personal/User Reservations - Enabling Reservations for End Users

In this topic:

By default, advance reservations are only available to scheduler administrators. While admins can create and manage reservations to provide resource access to end-users, end-users cannot create, modify, or destroy these reservations. Moab extends the ability to manage reservations to end-users and provides control facilities to keep these features manageable. Reservations created by end-users are called personal reservations or user reservations.

6.1.6.A Enabling Personal Reservation Management

User, or personal, reservations can be enabled on a per QoS basis by setting the ENABLEUSERRSV flag as in the following example:

```
QOSCFG[titan]    QFLAGS=ENABLEUSERRSV # allow 'titan' QOS jobs to create user
reservations
USERCFG[DEFAULT] QDEF=titan           # allow all users to access 'titan' QOS
...
```

If set, end-users are allowed to create, modify, cancel, and query reservations they own. As with jobs, users can associate a personal reservation with any QoS or account to which they have access. This is accomplished by specifying per reservation accountable credentials as in the following example:

```
> mrsvctl -c -S AQOS=titan -h node01 -d 1:00:00 -s 1:30:00
Note:  reservation test.126 created
```

As in the preceding example, a non-administrator user who wants to create a reservation must *ALWAYS* specify an accountable QoS with the mrsvctl -S flag. This specified QoS must have the `ENABLEUSERRSV` flag. By default, a personal reservation is created with an ACL of only the user who created it.

Example 6-8: Allow All Users in Engineering Group to Create Personal Reservations

```
QOSCFG[rsv]     QFLAGS=ENABLEUSERRSV # allow 'rsv' QOS jobs to create user
reservations
GROUPCFG[sales] QDEF=rsv             # allow all users in group sales to access 'rsv'
QOS
...
```

Example 6-9: Allow Specific Users to Create Personal Reservations

```
# special qos has higher job priority and ability to create user reservations
QOSCFG[special] QFLAGS=ENABLEUSERRSV
QOSCFG[special] PRIORITY=1000
# allow betty and steve to use the special qos
USERCFG[betty]  QDEF=special
USERCFG[steve]  QLIST=fast,special,basic QDEF=rsv
...
```

6.1.6.B Reservation Accountability

Personal reservations must be configured with a set of accountable credentials. These credentials (user, group, account, and so forth) indicate who is responsible for the resources dedicated by the reservation. If resources are dedicated by a reservation but not consumed by a job, these resources can be charged against the specified accountable credentials. Admins are allowed to create reservations and specify any accountable credentials for that reservation. While end-users can also be allowed to create and otherwise modify personal reservations, they are only allowed to create reservations with accountable credentials to which they have access. Further, while admins can manage any reservation, end-users can only control reservations they own.

Like jobs, reservation accountable credentials specify which credentials are charged for reservation usage and what policies are enforced as far as usage limits and allocation management is concerned. See 3.7.24 mrsvctl for more information on setting personal reservation credentials. While similar to jobs, personal reservations do have a separate set of usage limits and different allocation charging policies.

Setting Reservation Default Attributes

Organizations can use reservation profiles to set default attributes for personal reservations. These attributes can include reservation aspects such as management policies, charging credentials, ACLs, host constraints, and time frame settings.

6.1.6.C Reservation Limits

Allowing end-users the ability to create advance reservations can lead to potentially unfair and unproductive resource usage. This results from the fact that by default, there is nothing to prevent a user from reserving all resources in a given system or reserving resources during time slots that would greatly impede the scheduler's ability to schedule jobs efficiently. Because of this, it is highly advised that sites initially place either usage or allocation based constraints on the use of personal reservations. This can be achieved using Moab Accounting Manager (see the *Moab Accounting Manager Administrator Guide*).

6.1.6.D Reservation and Job Binding

Moab allows job-to-reservation binding to be configured at an admin or end-user level. This binding constrains how job to reservation mapping is allowed.

Constraining a Job to Only Run in a Particular Reservation

Jobs can be bound to a particular reservation at submit time (using the resource manager extension ADVRES) or dynamically using the mjobctl command (see 6.1.1.D Job to Reservation Binding). In either case, once bound to a reservation, a job can only run in that reservation even if other resources can be found outside of that reservation. The `mjobctl` command can also be used to dynamically release a job from reservation binding.

Example 6-10: Bind job to reservation

```
> mjobctl -m flags+=advres:grid.3 job1352
```

Example 6-11: Release job from reservation binding

```
> mjobctl -m flags-=advres job1352
```

Constraining a Reservation to Only Accept Certain Jobs

Binding a job to a reservation is independent of binding a reservation to a job. For example, a reservation may be created for user 'steve.' User 'steve' may then submit a number of jobs including one that is bound to that reservation using the `ADVRES` attribute. However, this binding simply forces that one job to use the reservation, it does not prevent the reservation from accepting other jobs submitted by user 'steve.' To prevent these other jobs from using the reserved resources, reservation to job binding must occur. This binding is accomplished by specifying either general job binding or specific job binding.

General job binding is the most flexible form of binding. Using the BYNAME attribute, a reservation can be created that only accepts jobs specifically bound to it.

Specific job binding is more constraining. This form of binding causes the reservation to only accept specific jobs, regardless of other job attributes and is set using the `JOB` reservation ACL.

Example 6-12: Configure a reservation to accept only jobs that are bound to it

```
> mrsvctl -m flags+=byname grid.3
```

Example 6-13: Remove general reservation to job binding

```
> mrsvctl -m flags-=byname grid.3
```

Example 6-14: Configure a reservation to accept a specific job

```
> mrsvctl -m -a JOB=3456 grid.3
```

Example 6-15: Remove a specific reservation to job binding

```
> mrsvctl -m -a JOB=3456 grid.3 --flags=unset
```

6.2 Partitions

In this section:

6.2.1 Partition Overview

Partitions are a logical construct that divide available resources. Any single resource (compute node) can only belong to a single partition. Often, natural hardware or resource manager bounds delimit partitions such as in the case of disjoint networks and diverse processor configurations within a cluster. For example, a cluster might consist of 256 nodes containing four 64 port switches. This cluster might receive excellent interprocess communication speeds for parallel job tasks located within the same switch but sub-stellar performance for tasks that span switches. To handle this, the site may choose to create four partitions, allowing jobs to run within any of the four partitions but not span them.

While partitions do have value, it is important to note that within Moab, the standing reservation facility provides significantly improved flexibility and should be used in the vast majority of politically motivated cases where partitions may be required under other resource management systems. Standing reservations provide time flexibility, improved access control features, and more extended resource specification options. Also, another Moab facility called Node Sets allows intelligent aggregation of resources to improve per job node allocation decisions. In cases where system partitioning is considered for such reasons, node sets may be able to provide a better solution.

Still, one key advantage of partitions over standing reservations and node sets is the ability to specify partition specific policies, limits, priorities, and scheduling algorithms although this feature is rarely required. An example of this need may be a cluster consisting of 48 nodes owned by the Astronomy Department and 16 nodes owned by the Mathematics Department. Each department may be willing to allow sharing of resources but wants to specify how their partition will be used. As mentioned, many of Moab's scheduling policies can be specified on a per partition basis allowing each department to control the scheduling goals within their partition.

The partition associated with each node should be specified as indicated in the section Node Location. With this done, partition access lists can be specified on a per job or per QoS basis to constrain which resources a job can have access to. See 6.3 Quality of Service (QoS) Facilities for more information. By default, QoSes and jobs allow global partition access. Note that by default, a job can only use resources within a single partition.

If no partition is specified, Moab creates one partition per resource manager into which all resources corresponding to that resource manager are placed. This partition is given the same name as the resource manager.

A partition cannot span multiple resource managers. In addition to these resource manager partitions, a pseudo-partition named '[ALL]' is created that contains the aggregate resources of all partitions.

While the resource manager partitions are real partitions containing resources not explicitly assigned to other partitions, the '[ALL]' partition is only a convenience object and is not a real partition; therefore it cannot be requested by jobs or included in configuration ACLs.

6.2.2 Defining Partitions

Node to partition mappings can be established directly using the NODECFG parameter or indirectly using the FEATUREPARTITIONHEADER parameter. If using direct mapping, this is accomplished as shown in the example below:

```
NODECFG[node001]   PARTITION=astronomy
NODECFG[node002]   PARTITION=astronomy
...
NODECFG[node049]   PARTITION=math
...
```

By default, Moab creates two partitions, 'DEFAULT' and '[ALL].' These are used internally, and consume spots in the 31-partition maximum defined in the `MMAX_PAR` parameter. If more partitions are needed, you can adjust the maximum partition count. See Appendix D: Adjusting Default Limits for information on increasing the maximum number of partitions.

6.2.3 Managing Partition Access

Partition access can be constrained by credential ACLs and by limits based on job resource requirements.

In this topic:

6.2.3.A Credential Based Access

Determining who can use which partition is specified using the *CFG parameters (USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, CLASSCFG, and SYSCFG). These parameters allow you to select a partition access list on a credential or system wide basis using the PLIST attribute. By default, the access associated with any given job is the logical OR of all partition access lists assigned to the job's credentials.

For example, assume a site with two partitions, general, and test. The site management wants everybody to use the general partition by default. However, one user, Steve, needs to perform the majority of his work on the test partition. Two special groups, staff and management will also need access to use the test partition from time to time but will perform most of their work in the general partition. The following example configuration enables the needed user and group access and defaults for this site:

```
SYSCFG[base]      PLIST=general:test
USERCFG[DEFAULT]  PLIST=general
USERCFG[steve]    PLIST=general:test
GROUPCFG[staff]   PLIST=general:test
GROUPCFG[mgmt]    PLIST=general:test
```

While using a logical OR approach allows sites to add access to certain jobs, some sites prefer to work the other way around. In these cases, access is granted by default and certain credentials are then restricted from accessing various partitions. To use this model, a system partition list must be specified as in the following example:

```
SYSCFG[base]     PLIST=general,test&
USERCFG[demo]    PLIST=test&
GROUPCFG[staff]  PLIST=general&
```

In the preceding example, note the ampersand (&). This character, which can be located anywhere in the PLIST line, indicates that the specified partition list should be logically ANDed with other partition access lists. In this case, the configuration limits jobs from user demo to running in partition test and jobs from group staff to running in partition general. All other jobs are allowed to run in either partition.

When using AND-based partition access lists, the base system access list must be specified with SYSCFG.

6.2.3.B Per Job Resource Limits

Access to partitions can be constrained based on the resources requested on a per job basis with limits on both minimum and maximum resources requested:

```
                    PARCFG[amd]     MAX.PROC=16
PARCFG[pIII]     MAX.WCLIMIT=12:00:00 MIN.PROC=4
PARCFG[aix]      MIN.NODE=12
```

All limits are specified using the parameter PARCFG. See 5.2.6 Usage-Based Limits for more information on the available limits.

6.2.4 Requesting Partitions

Users can request to use any partition they have access to on a per job basis. This is accomplished using the resource manager extensions since most native batch systems do not support the partition concept. For example, on a Torque system, a job submitted by a member of the group `staff` could request that the job run in the `test` partition by adding the line `-l partition=test` to the *qsub* command line. See 11.3 Resource Manager Extensions for more information on configuring and using resource manager extensions.

6.2.5 Per-Partition Settings

The following settings can be specified on a per-partition basis using the PARCFG parameter:

Setting	Description
FSSCALINGFACTOR	Moab will multiple the actual fairshare usage by this value to get the calculated fairshare usage of a job. The actual fairshare usage is calculated based on the FSPOLICY parameter. For an example, if FSPOLICY is set to DEDICATEDPS and a job runs on two processors for 100 seconds, then the actual fairshare usage is 200. If the job ran on a partition with FSSCALINGFACTOR=.5 then Moab multiplies 200*.5=100. If the job ran on a partition with FSSCALINGFACTOR=2 then Moab multiplies 200*2=400. `PARCFG[par1]    FSSCALINGFACTOR=<double>`
FSSECONDARYGROUPS	Map unix groups to fairshare groups.
GMETRIC	Specifies a generic metric to apply to the partition. It is configured like a Moab parameter, with the gmetric name inside square brackets. Specify multiple gmetrics by separating each configuration with a space. For example: `PARCFG[par1] GMETRIC[GM1]=20 GMETRIC[GM2]=10` *Partition `par1` has a `GM1` metric of 20 and a `GM2` metric of 10.*

Setting	Description
JOBNODEMATCHPOLICY	Specifies the JOBNODEMATCHPOLICY parameter to be applied to jobs that run in the specified partition.
NODEACCESSPOLICY	Specifies the NODEACCESSPOLICY parameter to be applied to jobs that run in the specified partition.
NODEALLOCATIONPOLICY	Specifies the NODEALLOCATIONPOLICY parameter to be applied to jobs that run in the specified partition.
RESOURCELIMITMULTIPLIER	Specifies the RESOURCELIMITMULTIPLIER[<PARID>] parameter to be applied to jobs that run in the specified partition. This can only be viewed with 'showconfig -v' `PARCFG[A] RESOURCELIMITMULTIPLER=PROC:1.1 RESOURCELIMITMULTIPLIER=MEM:2.0`

RESOURCELIMITPOLICY	Specifies the RESOURCELIMITPOLICY parameter to be applied to jobs that run in the specified partition.
	This can only be viewed with 'showconfig -v'
	`PARCFG[A] RESOURCELIMITPOLICY=WALLTIME:ALWAYS:CANCEL`
	`PARCFG[B] RESOURCELIMITPOLICY=WALLTIME:ALWAYS:REQUEUE`

6.2.6 Miscellaneous Partition Issues

A brief caution: Use of partitions has been quite limited in recent years as other, more effective approaches are selected for site scheduling policies. Consequently, some aspects of partitions have received only minor testing. Still, note that partitions are fully supported and any problem found will be rectified.

Related Topics

- 6.1.3 Standing Reservations
- 7.3 Node Sets
- FEATUREPARTITIONHEADER parameter
- PARCFG parameter

6.3 Quality of Service (QoS) Facilities

This section describes how to do the following:

- Allow key projects access to special services (such as preemption, resource dedication, and advance reservations).
- Provide access to special resources by requested QoS.
- Enable special treatment within priority and fairshare facilities by requested QoS.
- Provide exemptions to usage limits and other policies by requested QoS.
- Specify delivered service and response time targets.
- Enable job deadline guarantees.
- Control the list of QoSes available to each user and job.
- Enable special charging rates based on requested or delivered QoS levels.
- Enable limits on the extent of use for each defined QoS.
- Monitor current and historical usage for each defined QoS.

In this section:

6.3.1 QoS Overview

Moab's QoS facility enables you to give special treatment to various classes of jobs, users, groups, and so forth. Each QoS object can be thought of as a container of special privileges ranging from fairness policy exemptions, to special job prioritization, to special functionality access. Each QoS object also has an extensive access list of users, groups, and accounts that can access these privileges.

Sites can configure various QoSes each with its own set of priorities, policy exemptions, and special resource access settings. They can then configure user, group, account, and class access to these QoSes. A given job will have a default QoS and may have access to several additional QoSes. When the job is submitted, the submitter can request a specific QoS or just allow the default QoS to be used. Once a job is submitted, a user can adjust the QoS of the job at any time using the setqos command. The `setqos` command will only allow the user to modify the QoS of that user's jobs and only change the QoS to a QoS that this user has access to. Moab Admins can change the QOS of any job to any value.

Jobs can be granted access to QoS privileges if the QoS is listed in the system default configuration QDEF (QoS default) or QLIST (QoS access list), or if the QoS is specified in the QDEF or QLIST of a user, group, account, or class associated with that job. Alternatively, a user can access QoS privileges if that user is listed in the QoSes MEMBERULIST attribute.

The mdiag -q command can be used to obtain information about the current QoS configuration including specified credential access.

6.3.2 QoS Enabled Privileges

The privileges enabled via QoS settings can be broken into the following categories:

- Special Prioritization
- Service Access and Constraints
- Usage Limits and Overrides
- Service Access Thresholds
- Preemption Management

All privileges are managed via the QOSCFG parameter.

6.3.2.A Special Prioritization

Attribute	Description
FSTARGET	Specifies QoS fairshare target.
FSWEIGHT	Sets QoS fairshare weight offset affecting a job's fairshare priority component.
PRIORITY	Assigns priority to all jobs requesting particular QoS.
PROCWEIGHT	Sets QoS PROCWEIGHT weight offset affecting a job's resource priority component.
QTTARGET	Sets QoS queuetime target affecting a job's target priority component and QoS delivered.
QTWEIGHT	Sets QoS queuetime weight offset affecting a job's service priority component.

Attribute	Description
XFTARGET	Sets QoS XFactor target affecting a job's target priority component and QoS delivered.
XFWEIGHT	Sets QoS XFactor weight offset affecting a job's service priority component.

Example 6-16: QOSCFG

```
# assign priority for all qos geo jobs

QOSCFG[geo]   PRIORITY=10000
```

6.3.2.B Service Access and Constraints

The QoS facility can be used to enable special services and to disable default services. These services are enabled/disabled by setting the QoS `QFLAGS` attribute.

Flag	Description
DEADLINE	Job can request an absolute or relative completion deadline and Moab will reserve resources to meet that deadline. (An alternative priority based deadline behavior is discussed in the section PRIORITY FACTORS.)
DEDICATED	Moab dedicates all resources of an allocated node to the job meaning that the job will not share a node's compute resources with any other job.
ENABLEUSERRSV	Allow user or personal reservations to be created and managed.
IGNALL	Scheduler ignores all resource usage policies for jobs associated with this QoS.
JOBPRIOACCRUALPOLICY	Specifies how Moab should track the dynamic aspects of a job's priority. The two valid values are `ACCRUE` and `RESET`: • `ACCRUE` indicates that the job will accrue queuetime based priority from the time it is submitted unless it violates any of the policies not specified in JOBPRIOEXCEPTIONS. • `RESET` indicates that it will accrue priority from the time it is submitted unless it violates any of the JOBPRIOEXCEPTIONS. However, with `RESET`, if the job does violate JOBPRIOEXCEPTIONS then its queuetime based priority will be reset to 0.

Flag	Description
	JOBPRIOACCRUALPOLICY is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOACCRUALPOLICY=ACCRUE` The following old JOBPRIOACCRUALPOLICY values have been deprecated and should be adjusted to the following values: • QUEUEPOLICY = ACCRUE and JOBPRIOEXCEPTIONS SOFTPOLICY,HARDPOLICY • QUEUEPOLICYRESET = RESET and JOBPRIOEXCEPTIONS SOFTPOLICY,HARDPOLICY • ALWAYS = ACCRUE and JOBPRIOEXCEPTIONS ALL • FULLPOLICY = ACCRUE and JOBPRIOEXCEPTIONS NONE • FULLPOLICYRESET = RESET and JOBPRIOEXCEPTIONS NONE
JOBPRIOEXCEPTIONS	Specifies exceptions for calculating a job's dynamic priority (QUEUETIME, XFACTOR, TARGETQUEUETIME). Valid values are a comma-delimited list of any of the following: DEFER, DEPENDS, SOFTPOLICY, HARDPOLICY, IDLEPOLICY, USERHOLD, BATCHHOLD, and SYSTEMHOLD (ALL or NONE can also be specified on their own). Normally, when a job violates a policy, is placed on hold, or has an unsatisfied dependency, it will not accrue priority. Exceptions can be configured to allow a job to accrue priority in spite of any of these violations. With DEPENDS a job will increase in priority even if there exists an unsatisfied dependency. With SOFTPOLICY, HARDPOLICY, or IDLEPOLICY a job can accrue priority despite violating a specific limit. With DEFER, USERHOLD, BATCHHOLD, or SYSTEMHOLD a job can accrue priority despite being on hold. JOBPRIOEXCEPTIONS is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOEXCEPTIONS=IDLEPOLICY`
NOBF	Job is not considered for backfill.
NORESERVATION	Job should never reserve resources regardless of priority.
NTR	Job is prioritized as next to run (NTR) and backfill is disabled to

Flag	Description
	prevent other jobs from jumping in front of ones with the NTR flag. It is important to note that jobs marked with this flag should not be blocked. If they are, Moab will stop scheduling because if a job is marked with this flag, no other jobs will be run until the flagged NTR (Next to Run) job starts. Consider using the `PRIORITY` attribute of the QOSCFG[<QOSID>] parameter instead, when possible. Or, as you may encounter a scheduling delay for NTR-flagged jobs to start, consider using the parameters RESERVATIONDEPTH and RESERVATIONQOSLIST to provide better scheduling flow. See 6.1.4 Reservation Policies (especially the section on Assigning Per-QoS Reservation Creation Rules) for more information.
PREEMPTCONFIG	User jobs can specify options to alter how preemption impacts the job such as minpreempttime.
PREEMPTEE	Job can be preempted by higher priority `PREEMPTOR` jobs.
PREEMPTFSV	Job can be preempted by higher priority `PREEMPTOR` jobs if it exceeds its fairshare target when started.
PREEMPTOR	Job can preempt lower priority `PREEMPTEE` jobs.
PREEMPTSPV	Job can be preempted by higher priority `PREEMPTOR` jobs if it currently violates a soft usage policy limit.
PROVISION	If the job cannot locate available resources with the needed OS or software, the scheduler can provision a number of nodes to meet the needed OS or software requirements.
RESERVEALWAYS	Job should create resource reservation regardless of job priority.
RUNNOW	Boosts a job's system priority and makes the job a preemptor. RUNNOW overrides resource restrictions such as MAXJOB or MAXPROC.
TRIGGER	The job is able to directly specify triggers.
USERESERVED[:<RSVID>]	Job can only use resources within accessible reservations. If

Flag	Description
	`<RSVID>` is specified, job can only use resources within the specified reservation.

Example 6-17: For lowprio QoS job, disable backfill and make job preemptible

```
QOSCFG[lowprio]  QFLAGS=NOBF,PREEMPTEE
```

Example 6-18: Bind all jobs to chemistry reservation

```
QOSCFG[chem-b]  QFLAGS=USERESERVED:chemistry
```

Other QoS Attributes

In addition to the flags, there are attributes that alter service access.

Attribute	Description
SYSPRIO	Sets the system priority on jobs associated with this QoS. Example: All jobs submitted under a QoS sample receive a system priority of 1. `QOSCFG[sample] SYSPRIO=1` Once a system priority has been added to a job, either manually or through configuration, it can only be removed manually.
REQUESTGEOMETRY	Defines the size that is requested when Elastic Computing occurs. Potential values are 'PRIORITYJOBSIZE' or '<NODECOUNT>@<DURATION>'. If PRIORITYJOBSIZE is set, then the nodecount and duration for Elastic Computing is set in realtime to whatever is the size of the highest priority idle job. Example: `QOSCFG[sample] REQUESTGEOMETRY=12@4:00:00:00`

Per QoS Required Reservations

If desired, jobs associated with a particular QoS can be locked into a reservation or reservation group using the `REQRID` attribute. For example, to force jobs using QoS `jasper` to only use the resources within the `failsafe` standing reservation, use the following:

```
QOSCFG[jasper] REQRID=failsafe
...
```

6.3.2.C Usage Limits and Overrides

All credentials, including QoS, allow specification of job usage limits as described in 5.2.1.A Basic Fairness Policies. In such cases, jobs are constrained by the most limiting of all applicable policies. With QoSes, an override limit can also be specified and with this limit, jobs are constrained by the override, regardless of other limits specified. Using override limits, you can create custom QoSes that allow for more jobs or more processors.

Example 6-19: Override Limits

```
# staff QoS should have a limit of 48 jobs, ignoring the user limit
USERCFG[DEFAULT]   MAXJOB=10
QOSCFG[staff]      OMAXJOB=48
```

(See 5.2.2 Override Limits.)

The following parameters can override the throttling policies from other credentials:

Parameter	Description
OMAXJOB	Overrides a credential's limit on the number of jobs the credential can have active (starting or running) at any given time. Moab places a hold on all new jobs submitted by that credential once it has reached its maximum number of allowable jobs. Overrides a limit set using the MAXJOB parameter).
OMAXNODE	Overrides a credential's limit on the total number of compute nodes that can be in use by active jobs at any given time. Overrides a limit set using the MAXNODE parameter.
OMAXPE	Overrides a credential's limit on the total number of dedicated processor-equivalents the credential can have allocated by active jobs at any given time. Overrides a limit set using the MAXPE parameter.
OMAXPROC	Overrides a credential's limit on the total number of dedicated processors the credential can have allocated by active jobs at any given time. Overrides a limit set using the MAXPROC parameter.
OMAXPS	Overrides a credential's limit on the number of outstanding processor-seconds the credential can have allocated at any given time. Overrides a limit set using the MAXPS parameter.
OMAXJPROC	Overrides a limit on the total number of dedicated processors that can be allocated to an active job at a given time. Overrides limits set using `msub -W x=MAXPROC` or the `CLASSCFG MAX.PROC` attribute.
OMAXJPS	Limits the number of outstanding processor-seconds allocated to a job at any given time. Overrides a limit set using the `CLASSCFG MAX.PS` attribute.

Parameter	Description
`OMAXJWC`	Overrides a maximum wallclock limit per job. Overrides a limit set using the `CLASSCFG` `MAX.WCLIMIT` attribute.
`OMAXJNODE`	Overrides a limit on the total number of compute nodes that can be in use by a job at any given time. Overrides a limit set using the `CLASSCFG` `MAX.NODE` attribute.

6.3.2.D Service Access Thresholds

Jobs can be granted access to services such as preemption and reservation creation, and they can be granted access to resource reservations. However, with QoS thresholds, this access can be made conditional on the current queuetime and XFactor metrics of an idle job. The following table lists the available QoS service thresholds:

Threshold Attribute	Description
PREEMPTQTTHRESHOLD	A job with this QoS becomes a preemptor if the specified queuetime threshold is reached.
PREEMPTXFTHRESHOLD	A job with this QoS becomes a preemptor if the specified XFactor threshold is reached.
RSVQTTHRESHOLD	A job with this QoS can create a job reservation to guarantee resource access if the specified queuetime threshold is reached.
RSVXFTHRESHOLD	A job with this QoS can create a job reservation to guarantee resource access if the specified XFactor threshold is reached.
ACLQTTHRESHOLD	A job with this QoS can access reservations with a corresponding QoS ACL only if the specified queuetime threshold is reached.
ACLXFTHRESHOLD	A job with this QoS can access reservations with a corresponding QoS ACL only if the specified XFactor threshold is reached.
TRIGGERQTTHRESHOLD	If a job with this QoS fails to run before this threshold is reached, any failure triggers associated with this QoS will fire.

6.3.2.E QoS Metrics

Metric	Description
BACKLOGCOMPLETIONTIME	The estimated run-time to all idle jobs for a certain QoS. More specifically, it is the processor second count of all the idle jobs in the QOS, divided by the total processors on the system. `QQOSCFG[HIGH TRIGGER=EType=threshold,AType=exec,TType=elastic,threshold=BACKLOGCOMPLETIONTIME>1,Action="$HOME/geometry.pl blah",timeout=5:00` In order to calculate the BacklogCompletionTime, the QoS must have ENABLEPROFILING=TRUE, either on the QoS itself or on the DEFAULT QoS.

6.3.2.F Preemption Management

Job preemption facilities can be controlled on a per-QoS basis using the PREEMPTEE and PREEMPTOR flags. Jobs that are preemptible can optionally be constrained to only be preempted in a particular manner by specifying the QoS PREEMPTPOLICY attribute as in the following example:

```
QOSCFG[special] QFLAGS=PREEMPTEE PREEMPTPOLICY=CHECKPOINT
```

For preemption to be effective, a job must be marked as a preemptee and must be enabled for the requested preemption type. For example, if the PREEMPTPOLICY is set to suspend, a potential target job must be both a preemptee and marked with the job flag SUSPENDABLE. See 21.1.4 Suspending Jobs with Preemption for more information. If the target job is not suspendable, it will be either requeued or canceled. Likewise, if the PREEMPTPOLICY is set to requeue, the job will be requeued if it is marked restartable. Otherwise, it will be canceled.

The minimum time a job must run before being considered eligible for preemption can also be configured on a per-QoS basis using the PREEMPTMINTIME parameter, which is analogous to the JOBPREEMPTMINACTIVETIME. Conversely, PREEMPTMAXTIME sets a threshold for which a job is no longer eligible for preemption; see the parameter JOBPREEMPTMAXACTIVETIME for analogous details.

The PREEMPTEES attribute enables you to specify which QoSes that a job in a specific QoS is allowed to preempt. The PREEMPTEES list is a comma-delimited list of QoS IDs. When a PREEMPTEES attribute is specified, a job using that QoS can only preempt jobs using QoSes listed in the PREEMPTEES list. In turn, those QoSes must be flagged as PREEMPTEE as in the following example:

```
QOSCFG[a] QFLAGS=PREEMPTOR PREEMPTEES=b,c
```

```
QOSCFG[b] QFLAGS=PREEMPTEE
QOSCFG[c] QFLAGS=PREEMPTEE
```

In the example, jobs in the 'a' QoS can only preempt jobs in the b and c QoSes.

6.3.3 Managing QoS Access

In this topic:

6.3.3.A Specifying Credential Based QoS Access

You can define the privileges allowed within a QoS by using the QOSCFG parameter; however, in most cases access to the QoS is enabled via credential specific *CFG parameters, specifically USERCFG, GROUPCFG, ACCOUNTCFG, and CLASSCFG, which allow defining QoS access lists and QoS defaults. Specify credential specific QoS access by using the QLIST and/or QDEF attributes of the associated credential parameter.

6.3.3.B QOS Access via Logical OR

To enable QoS access, the QLIST and/or QDEF attributes of the appropriate user, group, account, or class/queue should be specified as in the following example:

```
# user john's jobs can access QOS geo, chem, or staff with geo as default
USERCFG[john]     QDEF=geo   QLIST=geo,chem,staff
# group system jobs can access the development qos
GROUPCFG[systems] QDEF=development
# class batch jobs can access the normal qos
CLASSCFG[batch]   QDEF=normal
```

By default, jobs can request a QoS if access to that QoS is allowed by any of the job's credentials. (In the previous example, a job from user john submitted to the class batch could request QoSes geo, chem, staff, or normal).

6.3.3.C QOS Access via Logical AND

If desired, QoS access can be masked or logically ANDed if the QoS access list is specified with a terminating ampersand (&) as in the following example:

```
# user john's jobs can access QOS geo, chem, or staff with geo as default
USERCFG[john]     QDEF=geo   QLIST=geo,chem,staff
```

```
# group system jobs can access the development qos
GROUPCFG[systems] QDEF=development
# class batch jobs can access the normal qos
CLASSCFG[batch]   QDEF=normal
# class debug jobs can only access the development or lowpri QoSes regardless of other
credentials
CLASSCFG[debug]   QLIST=development,lowpri&
```

Specifying QoS Based Access

QoS access can also be specified from within the QoS object using the QoS MEMBERULIST attribute as in the following example:

```
# define qos premiere and grant access to users steve and john
QOSCFG[premiere]  PRIORITY=1000  QFLAGS=PREEMPTOR  MEMBERULIST=steve,john
```

By default, if a job requests a QoS that it cannot access, Moab places a hold on that job. The parameter QOSREJECTPOLICY can be used to modify this behavior.

6.3.4 Requesting QoS Services at Job Submission

By default, jobs inherit a default QoS based on the user, group, class, and account associated with the job. If a job has access to multiple QoS levels, the submitter can explicitly request a particular QoS using the QoS resource manager extension as in the following example:

```
> msub -l nodes=1,walltime=100,qos=special3 job.cmd
```

Related Topics

- 2.7 Credentials
- 5.5 Accounting, Charging, and Allocation Management
- 6.1.5 Configuring and Managing Reservations - ROLLBACKOFFSET (rollback reservations)
- 9.9 Job Deadlines
- 21.1.6 Using QoS Preemption

Chapter 7: Optimizing Scheduling Behavior – Backfill and Node Sets

In this chapter:

7.1 Optimization

Moab optimizes cluster performance. Every policy, limit, and feature is designed to allow maximum scheduling flexibility while enforcing the required constraints. A driving responsibility of the scheduler is to do all in its power to maximize system use and to minimize job response time while honoring the policies that make up the site's mission goals.

However, as all jobs are not created equal, optimization must be abstracted slightly further to incorporate this fact. Cluster optimization must also focus on targeted cycle delivery. In the scientific HPC community, the true goal of a cluster is to maximize delivered research. For businesses and other organizations, the purposes may be slightly different, but all organizations agree on the simple tenet that the cluster should optimize the site's mission goals.

To obtain this goal, the scheduler has several levels of optimization it performs:

Level	Description
Workload Ordering	Prioritizing workload and utilizing backfill.
Intelligent Resource Allocation	Selecting those resources that best meet the job's needs or best enable future jobs to run (see 4.2 Node Allocation Policies).
Maximizing Intra-Job Efficiency	Selecting the type of nodes, collection of nodes, and proximity of nodes required to maximize job performance by minimizing both job compute and inter-process communication time (see 7.3 Node Sets and Node Allocation Policies).

Level	Description
Job Preemption	Preempting jobs to allow the most important jobs to receive the best response time (see Chapter 21: Preemption).
Utilizing Flexible Policies	Using policies that minimize blocking and resource fragmentation while enforcing needed constraints (see 5.2.4 Hard and Soft Limits and 6.1 Advance Reservations).

7.2 Backfill

In this section:

7.2.1 Backfill Overview

Backfill is a scheduling optimization that allows a scheduler to make better use of available resources by running jobs out of order. When Moab schedules, it prioritizes the jobs in the queue according to a number of factors and then orders the jobs into a highest priority first (or priority FIFO) sorted list. It starts the jobs one by one stepping through the priority list until it reaches a job it cannot start. Because all jobs and reservations possess a start time and a wallclock limit, Moab can determine the completion time of all jobs in the queue. Consequently, Moab can also determine the earliest the needed resources will become available for the highest priority job to start.

Backfill operates based on this earliest job start information. Because Moab knows the earliest the highest priority job can start, and which resources it will need at that time, it can also determine which jobs can be started without delaying this job. Enabling backfill allows the scheduler to start other, lower-priority jobs so long as they do not delay the highest priority job. If backfill is enabled, Moab protects the highest priority job's start time by creating a job reservation to reserve the needed resources at the appropriate time. Moab then can start any job that will not interfere with this reservation.

Backfill offers significant scheduler performance improvement. In a typical large system, enabling backfill increases system utilization by about 20% and improves turnaround time by an even greater amount. Because of the way it works, essentially filling in holes in node space, backfill tends to favor smaller and shorter running jobs more than larger and longer running ones. It is common to see over 90% of these small and short jobs backfilled. Consequently, sites will see marked improvement in the level of service delivered to the small, short jobs and moderate to little improvement for the larger, long ones.

With most algorithms and policies, there is a trade-off. Backfill is not an exception but the negative effects are minor. Because backfill locates jobs to run from throughout the idle job queue, it tends to diminish the influence of the job prioritization a site has chosen and therefore may negate some desired workload steering attempts through this prioritization. Although by default the start time of the highest priority job is protected by a reservation, there is nothing to prevent the third priority job from starting early and possibly delaying the start of the second priority job. This issue is addressed along with its trade-offs later in this section.

Another problem is a little more subtle. Consider the following scenario involving a two-processor cluster. Job A has a four-hour wallclock limit and requires one processor. It started one hour ago (time zero) and will reach its wallclock limit in three more hours. Job B is the highest priority idle job and requires two processors for one hour. Job C is the next highest priority job and requires one processor for two hours. Moab examines the jobs and correctly determines that job A must finish in three hours and therefore, the earliest job B can start is in three hours. Moab also determines that job C can start and finish in less than this amount of time. Consequently, Moab starts job C on the idle processor at time one. One hour later (time two), job A completes early. Apparently, the user overestimated the amount of time job A would need by a few hours. Since job B is now the highest priority job, it should be able to run. However, job C, a lower priority job was started an hour ago and the resources needed for job B are not available. Moab re-evaluates job B's reservation and determines that it can slide forward an hour. At time three, job B starts.

In review, backfill provided positive benefits. Job A successfully ran to completion. Job C was started immediately. Job B was able to start one hour sooner than its original target time, although, had backfill not been enabled, job B would have been able to run two hours earlier.

The scenario just described occurs quite frequently because user estimates for job duration are generally inaccurate. Job wallclock estimate accuracy, or wallclock accuracy, is defined as the ratio of wall time required to actually run the job divided by the wall time requested for the job. Wallclock accuracy varies from site to site but the site average is rarely better than 50%. Because the quality of the walltime estimate provided by the user is so low, job reservations for high priority jobs are often later than they need to be.

Although there do exist some minor drawbacks with backfill, its net performance impact on a site's workload is very positive. While a few of the highest priority jobs may get temporarily delayed, their position as highest priority was most likely accelerated by the fact that jobs in front of them were able to start earlier due to backfill. Studies have shown that only a very small number of jobs are truly delayed and when they are, it is only by a fraction of their total queue time. At the same time, many jobs are started significantly earlier than would have occurred without backfill.

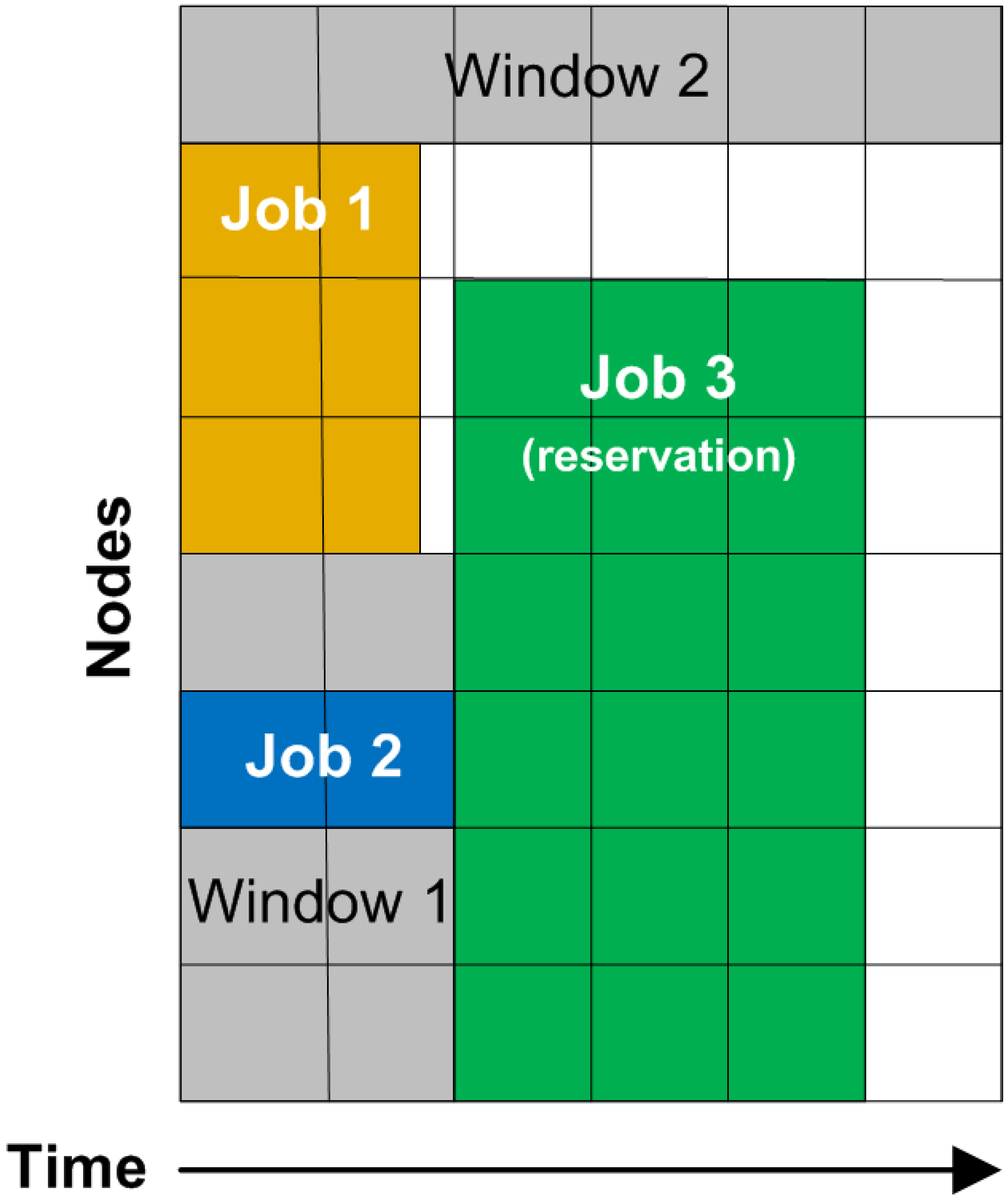
7.2.2 Backfill Algorithms

`BACKFILLPOLICY` controls which job gets selected first to be backfilled. Backfill jobs are still placed on nodes according to the parameter NODEALLOCATIONPOLICY.

The algorithm behind Moab backfill scheduling is straightforward, although there are a number of issues and parameters that should be highlighted. First of all, Moab makes two backfill scheduling passes. For each pass, Moab selects a list of jobs that are eligible for backfill. On the first pass, only those jobs that meet the constraints of the soft fairness throttling policies are considered and scheduled. The second pass expands this list of jobs to include those that meet the hard (less constrained) fairness throttling policies.

The second important concept regarding Moab backfill is the concept of backfill windows. The figure below shows a simple batch environment containing two running jobs and a reservation for a third job. The present time is represented by the leftmost end of the box with the future moving to the right. The light gray boxes represent currently idle nodes that are eligible for backfill. For this example, let's assume that the space represented covers 8 nodes and a 3 hour time frame. To determine backfill windows, Moab analyzes the idle nodes essentially looking for largest node-time rectangles. It determines that there are two backfill windows. The first window, Window 1, consists of 4 nodes that are available for only one hour (because some of the nodes are blocked by the reservation for Job 3). The second window contains only one node but has no time limit because this node is not blocked by the reservation for Job 3. It is important to note that these backfill windows overlap.

Image 7-1: Backfillable nodes create backfill windows 1 and 2



Once the backfill windows have been determined, Moab begins to traverse them. The current behavior is to traverse these windows widest window first (most nodes to fewest nodes). As each backfill window is evaluated, Moab applies the backfill algorithm specified by the BACKFILLPOLICY parameter.

If the `FIRSTFIT` algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that will actually fit in the current backfill window.
2. The first job is started.
3. While backfill jobs and idle resources remain, repeat step 1.

If the `BESTFIT` algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that actually fit in the current backfill window.
2. The degree of fit of each job is determined based on the BACKFILLMETRIC parameter (processors, seconds, processor-seconds).
3. The job with the best fit starts.
4. While backfill jobs and idle resources remain, repeat step 1.

If `NONE` is set, the backfill policy is disabled.

Other backfill policies behave in a generally similar manner. See Appendix A: Moab Parameters for more information.

Liberal Versus Conservative Backfill

By default, Moab reserves only the highest priority job resulting in a liberal and aggressive backfill. This reservation guarantees that backfilled jobs will not delay the highest priority job, although they might delay other jobs. The parameter RESERVATIONDEPTH controls how conservative or liberal the backfill policy is. This parameter controls how deep down the queue priority reservations will be made. While increasing this parameter improves guarantees that priority jobs will not be bypassed, it reduces the freedom of the scheduler to backfill resulting in somewhat lower system utilization. The significance of the trade-offs should be evaluated on a site by site basis.

7.2.3 Configuring Backfill

In this topic:

7.2.3.A Backfill Policies

Backfill is enabled in Moab by specifying the BACKFILLPOLICY parameter, which is used to control which job gets selected first to be backfilled. Once the job has been selected, it is still placed on nodes according to the NODEALLOCATIONPOLICY you have defined. By default, backfill is enabled in Moab using the `FIRSTFIT` algorithm. However, this parameter can also be set to `NONE` (disabled).

The number of reservations that protect the resources required by priority jobs can be controlled using the parameter RESERVATIONDEPTH. This depth can be distributed across job QoS levels using RESERVATIONQOSLIST.

7.2.3.B Backfill Chunking

In a batch environment saturated with serial jobs, serial jobs will, over time, dominate the resources available for backfill at the expense of other jobs. This is due to the time-dimension fragmentation associated with running serial jobs. For example, given an environment with an abundance of serial jobs, if a multi-processor job completes freeing processors, one of three things will happen:

1. The freed resources are allocated to another job requiring the same number of processors.
2. Additional jobs may complete at the same time allowing a larger job to allocate the aggregate resources.
3. The freed resources are allocated to one or more smaller jobs.

In environments where the scheduling iteration is much higher than the average time between completing jobs, case 3 occurs far more often than case 2, leading to smaller and smaller jobs populating the system over time.

To address this issue, the scheduler incorporates the concept of chunking. Chunking allows the scheduler to favor case 2 maintaining a more controlled balance between large and small jobs. The idea of chunking involves establishing a time-based threshold during which resources available for backfill are aggregated. This threshold is set using the parameter BFCHUNKDURATION. When resources are freed, they are made available only to jobs of a certain size (set using the parameter BFCHUNKSIZE) or larger. These resources remain protected from smaller jobs until either additional resources are freed up and a larger job can use the aggregate resources, or until the `BFCHUNKDURATION` threshold time expires.

Backfill chunking is only activated when a job of size `BFCHUNKSIZE` or larger is blocked in backfill due to lack of resources.

It is important to note that the optimal settings for these parameters is very site-specific and will depend on the workload (including the average job turnaround time, job size, and mix of large to small jobs), cluster resources, and other scheduling environmental factors. Setting too restrictive values needlessly reduces utilization while settings that are too relaxed do not allowed the desired aggregation to occur.

Backfill chunking is only enabled in conjunction with the `FIRSTFIT` backfill policy.

The current implementation of backfill chunking in Moab behaves as follows:

- When Moab starts, if backfill chunking is enabled, it starts immediately in a chunking window (with duration dictated by `BFChunkDuration`).
- Each time a job completes, the chunking window is re-enabled and extended to the chunking duration (current time + `BFChunkDuration`).
- While within this potentially continuously extending chunking window, the first backfill job, whether large or small, will always be evaluated without hindrance. Jobs will continue to be evaluated with the behavior that the first large job (based on `BFChunkSize`) to be encountered (including the first) will prevent subsequent small jobs from being backfilled.
- If a timeframe occurs where a job has not completed within `BFChunkDuration` of the previous job completion, then all job sizes can be freely backfilled.
- Backfill chunking does not kick in again until `BFChunkDuration` past the next job completion.

7.2.3.C Virtual Wallclock Time Scaling

In most environments, users submit jobs with rough estimations of the wallclock times. Within the HPC industry, a job typically runs for 40% of its specified wallclock time. Virtual Wallclock Time Scaling takes advantage of this fact to implement a form of optimistic backfilling. Jobs that are eligible for backfilling and not restricted by other policies are virtually scaled by the parameter BFVIRTUALWALLTIMESCALINGFACTOR (assuming that the jobs finish before this new virtual wallclock limit). The scaled jobs are then compared to backfill windows to see if there is space and time for them to be scheduled. The scaled jobs are only scheduled if there is no possibility that it will conflict with a standing or administrator reservation. Conflicts with such reservations occur if the virtual wallclock time overlaps a reservation, or if the original non-virtual wallclock time overlaps a standing or administrator reservation. Jobs that can fit into an available backfill window without having their walltime scaled are backfilled 'as-is' (meaning, without virtually scaling the original walltime).

> Virtual Wallclock Time Scaling is only enabled when the BFVIRTUALWALLTIMESCALINGFACTOR parameter is defined.

If a virtually-scaled job fits into a window, and is backfilled, it will run until completion or until it comes within one scheduling iteration (the parameter RMPOLLINTERVAL defines the exact time of an iteration) of the virtual wallclock time expiration. In the latter case the job's wallclock time is restored to its original time and Moab checks and resolves conflicts caused by this 'expansion.' Conflicts may occur when the backfilled job is restored to its full duration resulting in reservation overlap. The BFVIRTUALWALLTIMECONFLICTPOLICY parameter controls how Moab handles these conflicts.

If the `BFVIRTUALWALLTIMECONFLICTPOLICY` parameter is set to `NONE` or is not specified, the overlapped job reservations are rescheduled.

Related Topics

- BACKFILLDEPTH parameter
- BACKFILLPOLICY parameter
- Appendix A: Moab Parameters - BFMINVIRTUALWALLTIME
- 6.1.4 Reservation Policies

7.3 Node Sets

In this section:

7.3.1 Node Set Usage

While backfill improves the scheduler's performance, this is only half the battle. The efficiency of a cluster, in terms of actual work accomplished, is a function of both scheduling performance and individual job efficiency. In many clusters, job efficiency can vary from node to node, and with the node mix allocated. Most parallel jobs written in popular languages such as MPI or PVM do not internally load balance their workload and therefore run only as fast as the slowest node allocated. Consequently, these jobs run most effectively on homogeneous sets of nodes. However, while many clusters start out as homogeneous, they quickly evolve as new generations of compute nodes are integrated into the system. Research has shown that this integration, while improving scheduling performance due to increased scheduler selection, can actually decrease average job efficiency.

A feature called node sets allows jobs to request sets of common resources without specifying exactly what resources are required. Node set policy can be specified globally or on a per-job basis. In addition to their use in forcing jobs onto homogeneous nodes, these policies can also be used to guide jobs to one or more types of nodes on which a particular job performs best, similar to job preferences available in other systems. For example, an I/O intensive job might run best on a certain range of processor speeds, running slower on slower nodes, while wasting cycles on faster nodes. A job can specify `ANYOF:FEATURE:bigmem,fastos` to request nodes with the `bigmem` or `fastos` feature. Alternatively, if a simple feature-homogeneous node set is desired, `ONEOF:FEATURE` can be specified. On the other hand, a job may request a feature based node set with the configuration `ONEOF:FEATURE:bigmem,fastos`, in which case Moab will first attempt to locate adequate nodes where all nodes contain the `bigmem` feature. If such a set cannot be found, Moab will look for sets of nodes containing the other specified features. In highly heterogeneous clusters, the use of node sets improves job throughput by 10 to 15%.

Node sets can be requested on a system wide or per job basis. System wide configuration is accomplished via the `NODESET*` parameters while per job specification occurs via the resource manager extensions.

The GLOBAL node is included in all feature node sets.

When creating node sets, you have the option of using a fixed configuration or of creating node sets dynamically (by using the *`msub`* command). This section explains how to set up both node set use cases.

7.3.2 Node Set Configuration Examples

Global node sets are defined using the parameters NODESETPOLICY, NODESETATTRIBUTE, NODESETLIST, and NODESETISOPTIONAL. As stated before, you can create node sets dynamically (see the section Dynamic Example below) or with a fixed configuration (see the section Fixed Configuration Example below). The use of these parameters can be best highlighted with the two examples below.

In this topic:

7.3.2.A Fixed Configuration Example

In this example, a large site possesses a Myrinet based interconnect and wants to, whenever possible, allocate nodes within Myrinet switch boundaries. To accomplish this, they could assign node attributes to each node indicating which switch it was associated with (switchA, switchB, and so forth) and then use the following system wide node set configuration:

```
NODESETPOLICY      ONEOF
NODESETATTRIBUTE   FEATURE
NODESETISOPTIONAL  TRUE
NODESETLIST        switchA,switchB,switchC,switchD
...
```

Node Set Policy

In the preceding example, the NODESETPOLICY parameter is set to the policy ONEOF and tells Moab to allocate nodes within a single attribute set. Other node set policies are listed in the following table:

Policy	Description
ANYOF	Select resources from all sets contained in node set list. The job could span multiple node sets.
FIRSTOF	Select resources from first set to match specified constraints.
ONEOF	Select a single set that contains adequate resources to support job.

Node Set Attribute

The example's NODESETATTRIBUTE parameter is set to `FEATURE`, specifying that the node sets are to be constructed along node feature boundaries.

You could also set the `NODESETATTRIBUTE` to `VARATTR`, specifying that node sets are to be constructed according to VARATTR values on the job.

You could also set the `NODESETATTRIBUTE` to `RESERVATION`, specifying that node sets are to be constructed according to the reservations (or reservation groups) specified in the NODESETLIST parameter.

Node Set Constraint Handling

The next parameter, NODESETISOPTIONAL, indicates that Moab should not delay the start time of a job if the desired node set is not available but adequate idle resources exist outside of the set. Setting this parameter to `TRUE` basically tells Moab to attempt to use a node set if it is available, but if not, run the job as soon as possible anyway.

Setting `NODESETISOPTIONAL` to `FALSE` will force the job to always run in a complete nodeset regardless of any start delay this imposes.

Node Set List

Finally, the NODESETLIST value of `switchA switchB...` tells Moab to only use node sets based on the listed feature values. This is necessary since sites often use node features for many purposes and the resulting node sets would be of little use for switch proximity if they were generated based on irrelevant node features indicating things such as processor speed or node architecture.

To add nodes to the `NODESETLIST`, you must configure features on your nodes using the NODECFG FEATURES attribute:

```
NODECFG[node01] FEATURES=switchA
NODECFG[node02] FEATURES=switchA
NODECFG[node03] FEATURES=switchB
```

Nodes node01 and node02 contain the switchA feature, and node node03 contains the switchB feature.

Node Set Priority

When resources are available in more than one resource set, the NODESETPRIORITYTYPE parameter allows control over how the best resource set is selected. Values for this parameter are described in the following table:

Priority Type	Description	Details
AFFINITY	Avoid a resource set with negative affinity.	Choosing this type causes Moab to select a node set with no negative affinity nodes (nodes that have a reservation that with negative affinity). If all node sets have negative affinity, then Moab will select the first matching node set.
BESTFIT	Select the smallest resource set possible.	Choosing this type causes Moab, when selecting a node set, to eliminate sets that do not have all the required resources. From the remaining sets, Moab chooses the set with the least amount of resources. This priority type most closely matches the job requirements in order to waste the least amount of resources. This type minimizes fragmentation of larger resource sets.
FIRSTFIT	Select the first set with enough resources.	Moab will select the first nodeset with enough resources to satisfy the job. This is the fastest of the priority types.
MINLOSS	Select the resource set that results in the minimal wasted resources assuming no internal job load balancing is available. (Assumes parallel jobs only run as fast as the slowest allocated node.)	Choosing this type works only when using the following configuration: `NODESETATTRIBUTE FEATURE` In a `SHAREDMEM` environment (see G.2.1 Moab-NUMA-Support Integration Guide for more information), Moab will select the node set based on NUMA properties (the smallest feasible node set).
WORSTFIT	Select the largest resource set possible.	This type causes Moab, when choosing a node set, to eliminate sets that do not have all the required resources. From the remaining sets, Moab chooses the set with the greatest amount of resources. This type minimizes fragmentation of smaller resource sets, but increases fragmentation of larger resource sets.

7.3.2.B Dynamic Example

In this example, a site wants to be able to dynamically specify which `VARATTR` values the node set will be based on. To accomplish this, the following configuration in the `moab.cfg` file could be used:

```
NODESETISOPTIONAL FALSE
NODESETPOLICY     FIRSTOF
NODESETATTRIBUTE  VARATTR
```

Node Set Attribute

The example's NODESETATTRIBUTE parameter is set to `FEATURE`, specifying that the node sets are to be constructed along node feature boundaries.

You could also set the `NODESETATTRIBUTE` to `VARATTR`, specifying that node sets are to be constructed according to VARATTR values on the job.

You could also set the `NODESETATTRIBUTE` to `RESERVATION`, specifying that node sets are to be constructed according to the reservations (or reservation groups) specified in the NODESETLIST parameter.

Node Set Policy

In the preceding example, the NODESETPOLICY parameter is set to the policy `FIRSTOF` and tells Moab to allocate nodes from the first set that matches specified constraints.

Node Set Constraint Handling

The parameter, NODESETISOPTIONAL, indicates that Moab should not delay the start time of a job if the desired node set is not available but adequate idle resources exist outside of the set. Setting this parameter to `FALSE` will force the job to always run in a complete node set regardless of any start delay this imposes.

msub Example

With the configuration (above) set in the `moab.cfg`, Moab is configured for dynamic node sets. You can create node sets dynamically by using the msub -l command. For more information, see 11.3 Resource Manager Extensions. Use the following format:

`msub -l nodeset=FIRSTOF:VARATTR:<var>[=<value>],...`

For example, if you wanted to create a dynamic node set for the Provo datacenter:

```
msub -l nodeset=FIRSTOF:VARATTR:datacenter=Provo
```

This command causes Moab to set datacenter=Provo as the node set.

You can specify more than one VARATTR in the command. For example, if you want to create a dynamic node set for the Provo datacenter and the SaltLake datacenter:

```
msub -l nodeset=FIRSTOF:VARATTR:datacenter=Provo:datacenter=SaltLake
```

If you specify only `datacenter` (without specifying a value, such as `=Provo`), Moab will look up all possible values (values reported on the node for that VARATTR), and then choose one. So if, for example, you have nodes that have VARATTRs `datacenter=Provo`, `datacenter=SaltLake`, and `datacenter=StGeorge`, then specifying `msub -l nodeset=FIRSTOF:VARATTR:datacenter` will cause the job to run in Provo *or* SaltLake *or* StGeorge.

You should also note that Moab also adds the VARATTR (whether you specify it or if Moab chooses it) to the required attribute (REQATTR) of the job. For example, if you specify `datacenter=Provo` as the VARATTR, `datacenter=Provo` will also be added to the job REQATTR. Likewise, if you specify only `datacenter`, and Moab chooses `datacenter=SaltLake`, then `datacenter=SaltLake` will be added to the job REQATTR.

If you do not request a VARATTR in the nodeset of the *`msub -l`* command, the job will run as if it did not use node sets at all, and nothing will be added to its REQATTR.

If you manually specify a different REQATTR on a job (for example, `datacenter=SaltLake`) from the node set VARATTR (for example, `datacenter=Provo`), the job will never run.

7.3.3 NODESETPLUS

Moab supports additional NodeSet behavior by specifying the NODESETPLUS parameter. Possible values when specifying this parameter are `SPANEVENLY` and `DELAY`.

Neither `SPANEVENLY` nor `DELAY` will work with multi-req jobs or preemption.

Value	Description
SPANEVENLY	Moab attempts to fit all jobs within one node set, or it spans any number of node sets evenly. When a job specifies a NODESETDELAY, Moab attempts to contain the job within a single node set; if unable to do so, it spans node sets evenly, unless doing so would delay the job beyond the requested `NODESETDELAY`.
DELAY	Moab attempts to schedule the job within a nodeset for the configured NODESETDELAY. If Moab cannot find space for the job to start within `NODESETDELAY` (Moab considers future workload to determine if space will open up in time and might create a future reservation), then Moab schedules the job and ignores the nodeset requirement.

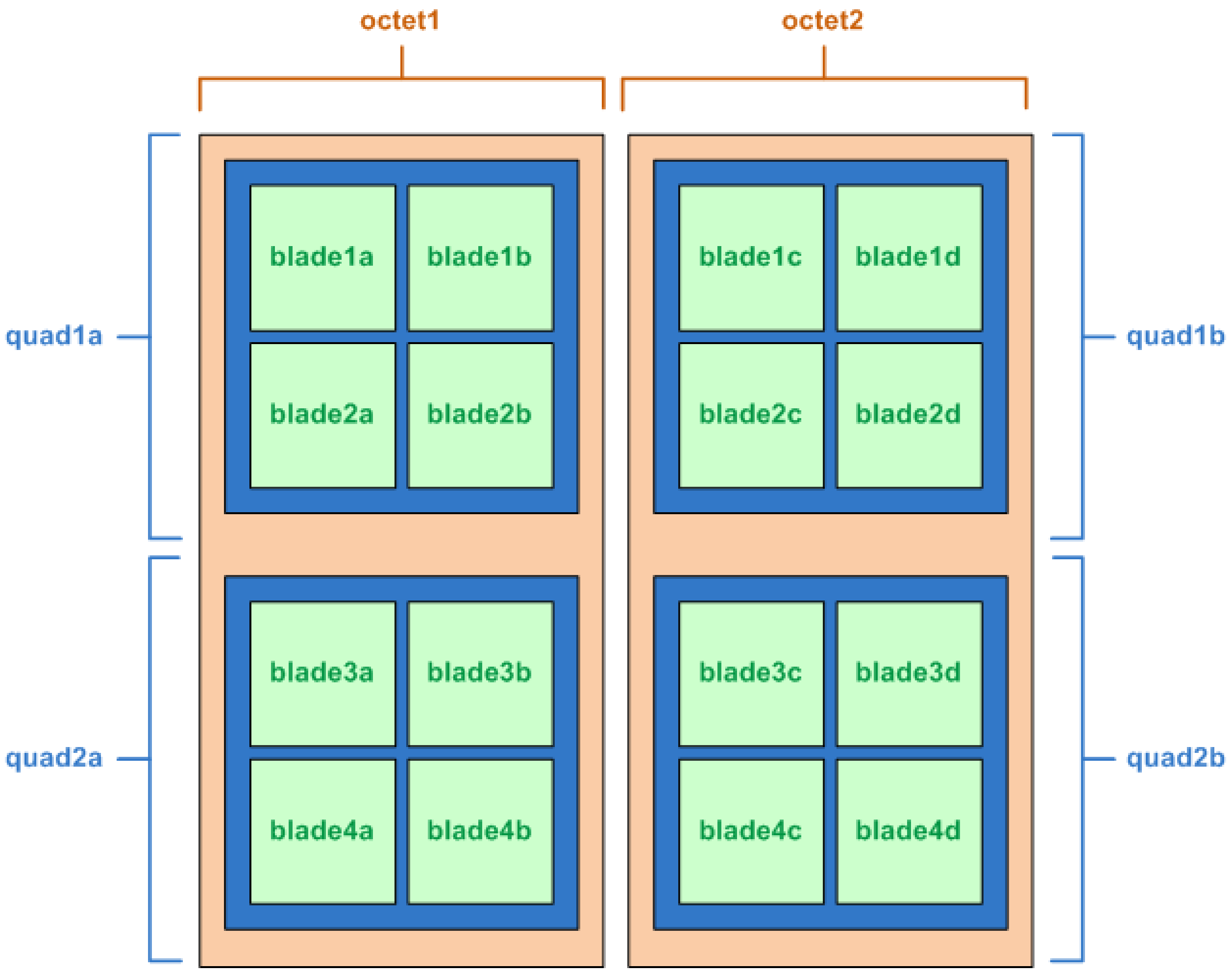
7.3.4 Nested Node Sets

Moab attempts to fit jobs on node sets in the order they are specified in the NODESETLIST. You can create nested node sets by listing your node sets in a specific order. Here is an example of a 'smallest to largest' nested node set:

```
NODESETPOLICY ONEOF
NODESETATTRIBUTE FEATURE
NODESETISOPTIONAL FALSE
NODESETLIST
blade1a,blade1b,blade2a,blade2b,blade3a,blade3b,blade4a,blade4b,quad1a,quad1b,quad2a,q
uad2b,octet1,octet2,sixteen
```

The accompanying cluster looks like this:

Image 7-2: Octet, quad, and blade node sets on a cluster



In this example, Moab tries to fit the job on the nodes in the blade sets first. If that doesn't work, it moves up to the nodes in the quad sets (a set of four blade sets). If the quads are insufficient, it tries the nodes in the octet sets (a set of four quad node sets).

7.3.5 Requesting Node Sets for Job Submission

On a per job basis, each user can specify the equivalent of all parameters except NODESETDELAY. As mentioned previously, this is accomplished using the resource manager extensions.

7.3.6 Configuring Node Sets for Classes

Classes can be configured with a default node set. In the configuration file, specify DEFAULT.NODESET with the following syntax: DEFAULT.NODESET=<SETTYPE>:<SETATTR>[:<SETLIST>[,<SETLIST>]...].

For example, in a heterogeneous cluster with two different types of processors, the following configuration confines jobs assigned to the amd class to run on either ATHLON or OPTERON processors:

```
CLASSCFG[amd] DEFAULT.NODESET=ONEOF:FEATURE:ATHLON,OPTERON
...
```

Related Topics

- 11.3 Resource Manager Extensions
- Parameter CLASSCFG[<CLASSID>]
- 6.2 Partitions

Chapter 8: Evaluating System Performance - Statistics, Profiling and Testing

In this chapter:

8.1 Moab Performance Evaluation

Moab Workload Manager tracks numerous performance statistics for jobs, accounting, users, groups, accounts, classes, QoS, the system, and so forth. These statistics can be accessed through various commands or Moab Cluster Manager/Monitor.

8.2 Accounting: Job and System Statistics

Moab provides extensive accounting facilities that allow resource usage to be tracked by resources (compute nodes), jobs, users, and other objects. The accounting facilities can be used in conjunction with, and correlated with, the accounting records provided by the resource and accounting manager.

Moab maintains both raw persistent data and a large number of processed in memory statistics allowing instant summaries of cycle delivery and system utilization. With this information, Moab can assist in accomplishing any of the following tasks:

- Determining cumulative cluster performance over a fixed time frame.
- Graphing changes in cluster utilization and responsiveness over time.
- Identifying which compute resources are most heavily used.
- Charting resource usage distribution among users, groups, projects, and classes.
- Determining allocated resources, responsiveness, and failure conditions for jobs completed in the past.
- Providing real-time statistics updates to external accounting systems.

This section describes how to accomplish each of these tasks using Moab tools and accounting information.

In this section:

8.2.1 Accounting Overview

Moab provides accounting data correlated to most major objects used within the cluster scheduling environment. These records provide job and reservation accounting, resource accounting, and credential-based accounting.

In this topic:

8.2.1.A Job and Reservation Accounting

As each job or reservation completes, Moab creates a complete persistent trace record containing information about who ran, the time frame of all significant events, and what resources were allocated. In addition, actual execution environment, failure reports, requested service levels, and other pieces of key information are also recorded. A complete description of each accounting data field can be found in 14.3 Workload Event Format.

8.2.1.B Resource Accounting

The load on any given node is available historically allowing identification of not only its usage at any point in time, but the actual jobs that were running on it. Moab Cluster Manager can show load information (assuming load is configured as a generic metric), but not the individual jobs that were running on a node at some point in the past. For aggregated, historical statistics covering node usage and availability, the *showstats* command can be run with the `-n` flag.

8.2.1.C Credential Accounting

Current and historical usage for users, groups, account, QoSes, and classes are determined in a manner similar to that available for evaluating nodes. For aggregated, historical statistics covering credential usage and availability, the *showstats* command can be run with the corresponding credential flag.

If needed, detailed credential accounting can also be enabled globally or on a credential by credential basis. With detailed credential accounting enabled, real-time information regarding per-credential usage over time can be displayed. To enable detailed per credential accounting, the ENABLEPROFILING attribute must be specified for credentials that are to be monitored. For example, to track detailed credentials, the following should be used:

```
USERCFG[DEFAULT]      ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]       ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]     ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]     ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]   ENABLEPROFILING=TRUE
```

Credential level profiling operates by maintaining a number of time-based statistical records for each credential. The parameters PROFILECOUNT and PROFILEDURATION control the number and duration of the statistical records.

8.2.2 Real-Time Statistics

Moab provides real-time statistical information about how the machine is running from a scheduling point of view. The showstats command is actually a suite of commands providing detailed information on an overall scheduling basis, and a per user, group, account and node basis. This command gets its information from in memory statistics that are loaded at scheduler start time from the scheduler checkpoint file. See 9.4 Checkpoint/Restart Facilities for more information. This checkpoint file is updated periodically and when the scheduler is shut down allowing statistics to be collected over an extended time frame. At any time, real-time statistics can be reset using the mschedctl -f command.

In addition to the showstats command, the showstats -f command also obtains its information from the in memory statistics and checkpoint file. This command displays a processor-time based matrix of scheduling performance for a wide variety of metrics. Information such as backfill effectiveness or average job queue time can be determined on a job size/duration basis.

FairShare Usage Statistics

Regardless of whether fairshare is enabled, detailed credential based fairshare statistics are maintained. Like job traces, these statistics are stored in the directory pointed to by the STATDIR parameter. Fairshare stats are maintained in a separate statistics file using the format `FS.<EPOCHTIME>` (`FS.982713600`, for example) with one file created per fairshare window. See 5.3 Fairshare for more information. These files are also flat text and record credential based usage statistics. Information from these files can be seen via the mdiag -f command.

Related Topics

- 10.5 Managing Shared Cluster Resources (Floating Resources)

8.3 Testing New Versions and Configurations

In this section:

8.3.1 MONITOR Mode

Moab supports a scheduling mode called `MONITOR`. In this mode, the scheduler initializes, contacts the resource manager and other peer services, and conducts scheduling cycles exactly as it would if running in `NORMAL` or production mode. Job are prioritized, reservations created, policies and limits enforced, and administrator and end-user commands enabled. The key difference is that although live resource management information is loaded, `MONITOR` mode disables Moab's ability to start, preempt, cancel, or otherwise modify jobs or resources. Moab continues to attempt to schedule exactly as it would in `NORMAL` mode but its ability to actually impact the system is disabled. Using this mode, you can quickly verify correct resource manager configuration and scheduler operation. This mode can also be used to validate new policies and constraints. In fact, Moab can be run in `MONITOR` mode on a production system while another scheduler or even another version of Moab is running on the same system. This unique ability can allow new versions and configurations to be fully tested without any exposure to potential failures and with no cluster downtime.

To run Moab in `MONITOR` mode, simply set the `MODE` attribute of the `SCHEDCFG` parameter to `MONITOR` and start Moab. Normal scheduler commands can be used to evaluate configuration and performance. Diagnostic commands can be used to look for any potential issues. Further, the Moab log file can be used to determine which jobs Moab attempted to start, and which resources Moab attempted to allocate.

If another instance of Moab is running in production and a site admin wants to evaluate an alternative configuration or new version, this is easily done but care should be taken to avoid conflicts with the primary scheduler. Potential conflicts include statistics files, logs, checkpoint files, and user interface ports. One of the easiest ways to avoid these conflicts is to create a new test directory with its own log and stats subdirectories. The new `moab.cfg` file can be created from scratch or based on the existing `moab.cfg` file already in use. In either case, make certain that the `PORT` attribute of the `SCHEDCFG` parameter differs from that used by the production scheduler by at least two ports. If testing with the production binary executable, the MOABHOMEDIR environment variable should be set to point to the new test directory to prevent Moab from loading the production `moab.cfg` file.

8.3.2 INTERACTIVE Mode

`INTERACTIVE` mode allows for evaluation of new versions and configurations in a manner different from `MONITOR` mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and asks for permission to complete it. For example, before starting a job, Moab will post something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The admin must specifically accept each command request after verifying it correctly meets desired site policies. Moab then executes the specified command. This mode is highly useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to `NORMAL` after verifying correct behavior.

Related Topics

- 14.1 Testing New Releases and Policies

Chapter 9: General Job Administration

In this chapter:

9.1 Job Holds

In this section:

9.1.1 Holds and Deferred Jobs

Moab supports job holds applied by users (user holds), administrators (system holds), and resource managers (batch holds). There is also a temporary hold known as a job defer.

9.1.2 User Holds

User holds are very straightforward. Many, if not most, resource managers provide interfaces by which users can place a hold on their own job that tells the scheduler not to run the job while the hold is in place. Users can use this capability because the job's data is not yet ready, or they want to be present when the job runs to monitor results. Such user holds are created by, and under the control of a non-privileged user and can be removed at any time by that user. As expected, users can only place holds on their jobs. Jobs with a user hold in place will have a Moab state of `Hold` or `UserHold` depending on the resource manager being used.

9.1.3 System Holds

The system hold is put in place by a system admin either manually or by way of an automated tool. As with all holds, the job is not allowed to run so long as this hold is in place. A batch admin can place and release system holds on any job regardless of job ownership. However, unlike a user hold, normal users cannot release a system hold even on their own jobs. System holds are often used during system maintenance and to prevent particular jobs from running in accordance with current system needs. Jobs with a system hold in place will have a Moab state of `Hold` or `SystemHold` depending on the resource manager being used.

9.1.4 Batch Holds

Batch holds are placed on a job by the scheduler itself when it determines that a job cannot run. The reasons for this vary but can be displayed by issuing the checkjob`<JOBID>` command. Possible reasons are included in the following list:

- No Resources — The job requests resources of a type or amount that do not exist on the system.
- System Limits — The job is larger or longer than what is allowed by the specified system policies.
- Bank Failure — The allocations bank is experiencing failures.
- No Allocations — The job requests use of an account that is out of allocations and no fallback account has been specified.
- RM Reject — The resource manager refuses to start the job.
- RM Failure — The resource manager is experiencing failures.
- Policy Violation — The job violates certain throttling policies preventing it from running now and in the future.
- No QOS Access — The job does not have access to the QoS level it requests.

Jobs that are placed in a batch hold will show up within Moab in the state `BatchHold`.

9.1.5 Job Defer

In most cases, a job violating these policies is not placed into a batch hold immediately; rather, it is deferred. The parameter DEFERTIME indicates how long it is deferred. At this time, it is allowed back into the idle queue and again considered for scheduling. If it again is unable to run at that time or at any time in the future, it is again deferred for the timeframe specified by DEFERTIME. A job is released and deferred up to DEFERCOUNT times at which point the scheduler places a batch hold on the job and waits for a system admin to determine the correct course of action. Deferred jobs have a Moab state of `Deferred`. As with jobs in the BatchHold state, the reason the job was deferred can be determined by use of the *`checkjob`* command.

At any time, a job can be released from any hold or deferred state using the releasehold command. The Moab logs should provide detailed information about the cause of any batch hold or job deferral.

Under Moab, the reason a job is deferred or placed in a batch hold is stored in memory but is not checkpointed. Therefore this information is available only until Moab is recycled at which point the checkjob command no longer displays this reason information.

Related Topics

- DEFERSTARTCOUNT parameter (number of job start failures allowed before job is deferred)

9.2 Job Priority Management

Job priority management is controlled via both configured and manual intervention mechanisms:

- Priority Configuration - see 4.1 Job Prioritization
- Manual Intervention - see 3.7.38.K setspri

9.3 Suspend/Resume Handling

When supported by the resource manager, Moab can suspend and resume jobs. By default, a job is suspended for one minute before it can be resumed by Moab. You can modify this default time using the MINADMINSTIME parameter.

Moab schedules suspended jobs each iteration to see if they can be resumed. If the node the jobs are running on is free, then Moab automatically resumes the job.

Alternatively, users can suspend their own jobs, but only an admin can resume them. The admin can resume jobs before the time set for Moab to resume.

suspendable

A job must be marked as `suspendable` for Moab to suspend and resume it. To do so, either submit the job with the `suspendable` flag attached to it or configure a credential to pass the flag to its associated jobs. These methods are demonstrated in the examples below:

```
msub -l flags=suspendable
```

```
GROUPCFG[default] JOBFLAGS=SUSPENDABLE
```

Once the job is suspendable, Moab enables you to suspend jobs using the two following methods: (1) manually on the command line and (2) automatically in the `moab.cfg` file.

mjobctl

To manually suspend jobs, use the mjobctl command as demonstrated in the following example:

```
> mjobctl -s job05
```

Moab suspends `job05`, preventing it from running immediately in the job queue.

If you are an admin and want to resume a job, use the *`mjobctl`* command as demonstrated in the following example:

```
> mjobctl -r job05
```

Moab removes `job05` from a suspended state and allows it to run.

SUSPEND

You can also configure the Moab preemption policy to suspend and resume jobs automatically by setting the PREEMPTPOLICY parameter to `SUSPEND`. A sample Moab configuration looks like this:

```
PREEMPTPOLICY SUSPEND
...
USERCFG[tom] JOBFLAGS=SUSPENDABLE
```

Moab suspends jobs submitted by user `tom` if necessary to make resources available for jobs with higher priority.

If your resource manager has a native interface, you must configure the parameter JOBSUSPENDURL to suspend and resume jobs.

For more information about suspending and resuming jobs in Moab, see the following sections:

- 3.7.20 mjobctl - manual preemption
- Chapter 21: Preemption

9.4 Checkpoint/Restart Facilities

Checkpointing records the state of a job, allowing for it to restart later without interruption to the job's execution. Checkpointing can be performed manually, as the result of triggers or events, or in conjunction with various QoS policies.

Moab's ability to checkpoint is dependent upon both the cluster's resource manager and operating system. In most cases, two types of checkpoint are enabled, including (1) checkpoint and continue and (2) checkpoint and terminate. While either checkpointing method can be activated using the mjobctl command, only the checkpoint and terminate type is used by internal scheduling and event managements facilities.

Checkpointing behavior can be configured on a per-resource manager basis using various attributes of the RMCFG parameter.

Related Topics

- Chapter 21: Preemption
- PREEMPTPOLICY parameter
- Resource Manager CHECKPOINTSIG attribute
- Resource Manager CHECKPOINTTIMEOUT attribute

9.5 Job Dependencies

In this section:

9.5.1 Basic Job Dependency Support

By default, basic single step job dependencies are supported through completed/failed step evaluation. Basic dependency support does not require special configuration and is activated by default. Dependent jobs are only supported through a resource manager and therefore submission methods depend upon the specific resource manager being used.

Use the `-l depend=<STRING>` flag for the Torque `qsub` command and the Moab `msub` command. See the resource manager extension `DEPEND` value for more information.

Torque `qsub` also supports the `-W x=depend=<STRING>` or `-W depend=<STRING>` flag. Moab `msub` command also supports the `-W x=depend=<STRING>` flag.

Dependencies submitted with `-W` are handled by Torque, while dependencies submitted with `-l` are handled by Moab. For array dependencies, if any subjob fails then the status of the entire job array will be marked with a failure and any `afterok` dependencies will not be satisfied. Moab does not use the Torque `afteranyarray/afterokarray` syntax but instead uses the `after/afterok` syntax for both normal jobs and job arrays.

For other resource managers, consult the resource manager specific documentation.

9.5.2 Job Dependency Syntax

Dependency	Format	Description
after	`after:<job>[:<job>]...`	Job can start at any time after specified jobs have started execution.
afterany	`afterany:<job>[:<job>]...`	Job can start at any time after all specified jobs have completed regardless of completion status.
afterok	`afterok:<job>[:<job>]...`	Job can start at any time after all specified jobs have successfully completed.

Dependency	Format	Description
afternotok	`afternotok:<job>[:<job>]...`	Job can start at any time after all specified jobs have completed unsuccessfully.
before	`before:<job>[:<job>]...`	Job can start at any time before specified jobs have started execution.
beforeany	`beforeany:<job>[:<job>]...`	Job can start at any time before all specified jobs have completed regardless of completion status.
beforeok	`beforeok:<job>[:<job>]...`	Job can start at any time before all specified jobs have successfully completed.
beforenotok	`beforenotok:<job>[:<job>]...`	Job can start at any time before any specified jobs have completed unsuccessfully.
on	`on:<count>`	Job can start after `<count>` dependencies on other jobs have been satisfied.
synccount	`synccount:<count>`	Job is the first in a set of jobs to be executed at the same time. `<count>` is the number of additional jobs in the set, which can be up to 5. synccount is valid for single-request jobs with Torque as the resource manager.
syncwith	`syncwith:<job>`	Job is an additional member of a set of jobs to be executed at the same time. Moab supports up to 5 jobs. syncwith is valid for single-request jobs with Torque as the resource manager.

`<job>={jobname.jobname|jobid}`

When using JobName dependencies, prepend 'jobname.' to avoid ambiguity.

The `before*`, `synccount`, and `syncwith` dependencies do not work with jobs submitted with *`msub`*; they work only with *`qsub`*.

`before*` - we do not recommend using these dependencies because the job numbers of follow-up jobs would not be known yet.

Any of the dependencies containing `before` must be used in conjunction with the `on` dependency. So, if job A must run before job B, job B must be submitted with `depend=on:1`, and job A having `depend=before:A`. This means job B cannot run until one dependency of another job on job B has been fulfilled. This prevents job B from running until job A can be successfully submitted.

When you submit a dependency job and the dependency is not met, the job will remain idle in the queue indefinitely. To configure Moab to automatically cancel these failed dependency jobs, set the CANCELFAILEDDEPENDENCYJOBS scheduler flag. Moab also lets you cancel all jobs that a specified `<job_id>` depends on using `mjobctl -c flags=follow-dependency <job_id>`.

Related Topics

- 9.9 Job Deadlines

9.6 Job Defaults and Per Job Limits

In this section:

9.6.1 Job Defaults

Job defaults can be specified on a per queue basis. These defaults are specified using the CLASSCFG parameter. The following table shows the applicable attributes:

Attribute	Format	Example
DEFAULT.FEATURES	comma-delimited list of node features	`CLASSCFG[batch] DEFAULT.FEATURES=fast,io` *Jobs submitted to class `batch` will request nodes features `fast` and `io`.*
DEFAULT.WCLIMIT	`[[[DD:]HH:]MM:]SS`	`CLASSCFG[batch] DEFAULT.WCLIMIT=1:00:00` *Jobs submitted to class `batch` will request one hour of walltime by default.*

9.6.2 Per Job Maximum Limits

Job maximum limits can be specified on a per queue basis. These defaults are specified using the CLASSCFG parameter. The following table shows the applicable attributes:

Attribute	Format	Example
MAX.WCLIMIT	`[[[DD:]HH:]MM:]SS`	`CLASSCFG[batch] MAX.WCLIMIT=1:00:00` *Jobs submitted to class* `batch` *can request no more than one hour of walltime.*

9.6.3 Per Job Minimum Limits

Furthermore, minimum job defaults can be specified with the CLASSCFG parameter. The following table shows the applicable attributes:

Attribute	Format	Example
MIN.PROC	`<integer>`	`CLASSCFG[batch] MIN.PROC=10` *Jobs submitted to class* `batch` *can request no less than ten processors.*

Related Topics

- 5.2.6 Usage-Based Limits

9.7 General Job Policies

In this section:

There are a number of configurable policies that help control advanced job functions. These policies help determine allowable job sizes and structures.

9.7.1 Multi-Node Support

You can configure the ability to allocate resources from multiple nodes to a job with the MAX.NODE limit.

9.7.2 Multi-Req Support

Jobs can specify multiple types of resources for allocation. For example, a job could request 4 nodes with 256 MB of memory and 8 nodes with feature `fast` present.

Resources specified in a multi-req job are delimited with a plus sign (+).

Neither `SPANEVENLY` nor `DELAY` values of the NODESETPLUS parameter will work with multi-req jobs or preemption.

Example 9-1:

This requests 4 nodes with 1 proc each, 10 nodes with 5 procs each, and 2 nodes with 2 procs each:

```
-l nodes=4:ppn=1+10:ppn=5+2:ppn=2
```

The total number of processors requested is (4*1) + (10*5) + (2*2), or 58 processors.

Example 9-2:

The job submitted requests a total of 16 nodes:

```
-l nodes=15+1:ppn=4
```

15 of these nodes have no specific requirements, but the remaining node must have 4 processors.

Example 9-3:

The job requests a total of 4 nodes; 3 nodes with the `fast` feature and 1 node with the `io` feature:

```
-l nodes=3:fast+1:io
```

9.7.3 Malleable Job Support

A job can specify whether it is able to use more processors or less processors and what effect, if any, that has on its wallclock time. For example, a job may run for 10 minutes on 1 processor, 5 minutes on 2 processors and 3 minutes on 3 processors. When a job is submitted with a task request list attached, Moab determines which task request fits best and molds the job based on its specifications. To submit a job with a task request list and allow Moab to mold it based on the current scheduler environment, use the TRL (Format 1) or the TRL (Format 2) flag in the Resource Manager Extension.

9.7.4 Enabling Job User Proxy

By default, user proxying is disabled. To be enabled, it must be authorized using the `PROXYLIST` attribute of the USERCFG parameter. This parameter can be specified either as a comma-delimited list of users or as the keyword `validate`. If the keyword `validate` is specified, the RMCFG attribute `JOBVALIDATEURL` should be set and used to confirm that the job's owner can proxy to the job's execution user. An example script performing this check for ssh-based systems is provided in the `tools` directory (see the JOBVALIDATEURL tool).

For some resource managers, proxying must also be enabled at the resource manager level. The following example shows how ssh-based proxying can be accomplished in a Moab+Torque with SSH environment.

To validate proxy users, Moab must be running as root.

Example 9-4: SSH Proxy Settings

```
USERCFG[DEFAULT] PROXYLIST=validate
RMCFG[base]  TYPE=<resource manager>
JOBVALIDATEURL=exec://$HOME/tools/job.validate.sshproxy.pl
```

```
> qmgr -c 's s allow_proxy_user=true'
> su - testuser
> qsub -I -u testuser2
qsub: waiting for job 533.igt.org to start
qsub: job 533.igt.org ready
testuser2@igt:~$
```

In this example, the validate tool, '`job.validate.sshproxy.pl`', can verify proxying is allowed by becoming the submit user and determining if the submit user can achieve passwordless access to the specified execution user. However, site-specific tools can use any method to determine proxy access including a flat file look-up, database lookup, querying of an information service such as NIS or LDAP, or other local or remote tests. For example, if proxy validation is required but end-user accounts are not available on the management node running Moab, the job validate service could perform the validation test on a representative remote host such as a login host.

This feature supports `qsub` only.

The JOBVALIDATEURL tool is highly flexible allowing any combination of job attributes to be evaluated and tested using either local or remote validation tests. The validate tool allows not only pass/fail responses but also allows the job to be modified, or rejected in a custom manner depending on the site or the nature of the failure.

Related Topics

- 5.2 Usage Limits/Throttling Policies
- qmgr -c in the *Torque Resource Manager Administrator Guide*
- qsub -I in the *Torque Resource Manager Administrator Guide*

9.8 Using a Local Queue

Moab allows jobs to be submitted directly to the scheduler. With a local queue, Moab is able to directly manage the job or translate it for resubmission to a standard resource manager queue. There are multiple advantages to using a local queue:

- Jobs can be translated from one resource manager job submission language to another (such as submitting a PBS job and running it on a Loadleveler cluster).
- Jobs can be migrated from one local resource manager to another.
- Jobs can be migrated to remote systems using Moab peer-to-peer functionality.
- Jobs can be dynamically modified and optimized by Moab to improve response time and system utilization.
- Jobs can be dynamically modified to account for system hardware failures or other issues.
- Jobs can be dynamically modified to conform to site policies and constraints.
- Grid jobs are supported.

Local Queue Configuration

A local queue is configured just like a standard resource manager queue. It can have defaults, limits, resource mapping, and credential access constraints. The following table describes the most common settings:

Default queue	
Format	`RMCFG[internal] DEFAULTCLASS=<CLASSID>`
Description	The job class/queue assigned to the job if one is not explicitly requested by the submitter. All jobs submitted directly to Moab are initially received by the pseudo-resource manager `internal`. Therefore, default queue configuration can only be applied to it.
Example	`RMCFG[internal] DEFAULTCLASS=batch`

Class default resource requirements	
Format	`CLASSCFG[<CLASSID>] DEFAULT.FEATURES=<X> CLASSCFG [<CLASSID>] DEFAULT.MEM=<X> CLASSCFG[<CLASSID>] DEFAULT.NODE=<X> CLASSCFG[<CLASSID>] DEFAULT.NODESET=<X> CLASSCFG[<CLASSID>] DEFAULT.PROC=<X> CLASSCFG[<CLASSID>] DEFAULT.WCLIMIT=<X>`
Description	The settings assigned to the job if not explicitly set by the submitter. Default values are available for node features, per task memory, node count, nodeset configuration, processor count, and wallclock limit.

Example	`CLASSCFG[batch] DEFAULT.WCLIMIT=4 DEFAULT.FEATURES=matlab`
	or
	`CLASSCFG[batch] DEFAULT.WCLIMIT=4`
	`CLASSCFG[batch] DEFAULT.FEATURES=matlab`

Class maximum resource limits	
Format	`CLASSCFG[<CLASSID>] MAX.FEATURES=<X> CLASSCFG [<CLASSID>] MAX.NODE=<X> CLASSCFG[<CLASSID>] MAX.PROC=<X> CLASSCFG[<CLASSID>] MAX.WCLIMIT=<X>`
Description	The maximum node features, node count, processor count, and wallclock limit allowed for a job submitted to the class/queue. If these limits are not satisfied, the job is not accepted and the submit request fails. `MAX.FEATURES` indicates that only the listed features can be requested by a job.

Class maximum resource limits	

Example	`CLASSCFG[smalljob] MAX.PROC=4 MAX.FEATURES=slow,matlab`
	or
	`CLASSCFG[smalljob] MAX.PROC=4`
	`CLASSCFG[smalljob] MAX.FEATURES=slow,matlab`

Class minimum resource limits	
Format	`CLASSCFG[<CLASSID>] MIN.FEATURES=<X> CLASSCFG [<CLASSID>] MIN.NODE=<X> CLASSCFG[<CLASSID>] MIN.PROC=<X> CLASSCFG[<CLASSID>] MIN.WCLIMIT=<X>`
Description	The minimum node features, node count, processor count, and wallclock limit allowed for a job submitted to the class/queue. If these limits are not satisfied, the job is not accepted and the submit request fails. `MIN.FEATURES` indicates that only the listed features can be requested by a job.

Example	`CLASSCFG[bigjob] MIN.PROC=4 MIN.WCLIMIT=1:00:00`
	or
	`CLASSCFG[bigjob] MIN.PROC=4`
	`CLASSCFG[bigjob] MIN.WCLIMIT=1:00:00`

Class access	
Format	`CLASSCFG[<CLASSID>] REQUIREDUSERLIST=<USERID> [,<USERID>]...`
Description	The list of users who can submit jobs to the queue.
Example	`CLASSCFG[math] REQUIREDUSERLIST=john,steve`

Available resources	
Format	`CLASSCFG[<CLASSID>] HOSTLIST=<HOSTID>[,<HOSTID>]...`
Description	The list of nodes that jobs in the queue can use.
Example	`CLASSCFG[special] HOSTLIST=node001,node003,node13`

Class mapping between multiple sites is described in the section on Moab grid facilities.

If a job is submitted directly to the resource manager used by the local queue, the class default resource requirements are not applied. Also, if the job violates a local queue limitation, the job is accepted by the resource manager, but placed in the Blocked state.

9.9 Job Deadlines

In this section:

9.9.1 Deadline Overview

Job deadlines can be specified on a per job and per credential basis and are also supported using both absolute and QoS based specifications. A job requesting a deadline is first evaluated to determine if the deadline is acceptable. If so, Moab adds it to the list of deadline jobs and allocates resources to guarantee that all accepted deadline jobs are able to complete on or before their requested deadline. Once the scheduler confirms that all deadlines can be satisfied, it then optimizes resource allocation (in priority order) attempting to execute all jobs at the earliest possible time.

9.9.2 Setting Job Deadlines via QoS

Two types of job deadlines exist in Moab. The priority-based deadline linearly increases a job's priority as its deadline approaches (see the section Deadline (DEADLINE) Subcomponent for more information). The QoS method enables you to set a job completion time on job submission if, and only if, it requests and is allowed to access a QoS with the `DEADLINE` QFLAG set. This method is more powerful than the priority method, because Moab will attempt to make a reservation for the job as soon as the job enters the queue in order to meet the deadline, essentially bumping it to the front of the queue.

When a job is submitted to a QoS with the `DEADLINE` flag set, the job's `-l deadline` attribute is honored. If such QoS access is not available, or if resources do not exist at job submission time to allow the deadline to be satisfied, the job's deadline request is ignored.

Two methods exist for setting deadlines with a QoS:

- Submitting a job to a deadline-enabled QoS and specifying a deadline using *msub -l*.
- Submitting a job to a deadline-enabled QoS with a QTTARGET specified.

9.9.2.A Setting Job Deadlines at Job Submission

This method of setting a job deadline enables you to specify a job deadline as you submit the job. You can set the deadline as either an exact date and time or as an amount of time after job submission (i.e., three hours after submission).

To specify a deadline on job submission

1. In `moab.cfg`, create a QoS with the `DEADLINE` flag enabled:

   ```
   ...
   QOSCFG[special] QFLAGS=DEADLINE
   ```

   Jobs requesting the QoS `special` may submit jobs with a deadline that Moab will honor.

2. Submit a job to the QoS and set a deadline. This can be either absolute or relative.

   a. For an absolute deadline, use the format `hh:mm:ss_mm/dd/yy`. The following configuration sets a deadline for a job to finish by 8 A.M. on March 15th, 2025:

      ```
      msub -l qos=special deadline=08:00:00_03/15/25 job.sh
      ```

      The job must finish running by 8 A.M. on March 15, 2025.

   b. For a relative deadline, or the completion deadline of the job relative to its submission time, use the time format `[[[DD:]HH:]MM:]SS`:

      ```
      msub -l qos=special deadline=5:00:00 job.sh
      ```

      The job's deadline is 5 hours after its submission.

9.9.2.B Submitting a Job to a QoS with a Preconfigured Deadline

You can also set a relative job deadline by limiting the job's queue time. This method enables you to pre-configure the deadline rather than giving the power to specify a deadline to the user submitting the job. For jobs requesting these QoSes, Moab identifies and sets job deadlines to satisfy the corresponding response time targets.

To submit a job to a QoS with a preconfigured deadline

1. In `moab.cfg`, create a QoS with both the `DEADLINE` `QFLAG` and a response time target (`QTTARGET`). The `QTTARGET` is the maximum amount of time that Moab should allow the job to be idle in the queue.

   ```
   ...
   QOSCFG[special2] QFLAGS=DEADLINE QTTARGET=1:00:00
   ```

   Given this configuration, a job requesting QoS `special2` must spend a maximum of one hour in the queue.

2. Submit a job requesting the `special2` quality of service:

   ```
   msub -l qos=special2 walltime=2:00:00 job.sh
   ```

   This two-hour job has a completion time deadline set to three hours after its submission (one hour of target queue time and two hours of run time).

9.9.3 Job Termination Date

In addition to job completion targets, jobs can also be submitted with a TERMTIME attribute. The scheduler attempts to complete the job prior to the termination date, but if it is unsuccessful, it will terminate (cancel) the job once the termination date is reached.

9.9.4 Conflict Policies

The specific policy can be configured using the DEADLINEPOLICY parameter. Moab does not have a default policy for this parameter.

Policy	Description
CANCEL	The job is canceled and the user is notified that the deadline could not be satisfied.
HOLD	The job has a batch hold placed on it indefinitely. The admin can then decide what action to take.
RETRY	The job continually retries each iteration to meet its deadline; note that when used with `QTTARGET` the job's deadline continues to slide with relative time.
IGNORE	The job has its request ignored and is scheduled as normal.

Deadline scheduling might not function properly with per partition scheduling enabled. Check that `PARALLOCATIONPOLICY` is disabled to ensure `DEADLINEPOLICY` will work correctly.

Related Topics

- 6.3 Quality of Service (QoS) Facilities
- Job Submission Eligible Start Time constraints

9.10 Job Arrays

In this section:

9.10.1 Job Array Overview

You can submit an array of jobs to Moab via the msub command. Array jobs are an easy way to submit many subjobs that perform the same work using the same script, but operate on different sets of data. Subjobs are the jobs created by a job array and are identified by the job array ID and an index; for example, if `235[1]` is an identifier, the number `235` is a job array ID, and `1` is the subjob.

Subjobs of an array are executed in subjob index order.

Moab job arrays are different from Torque job arrays.

9.10.2 Enabling Job Arrays

To enable job arrays, include the ENABLEJOBARRAYS parameter in the Moab configuration file (`moab.cfg`).

9.10.3 Subjob Definitions

Like a normal job, an array job submits a job script, but it additionally has a start index (`sidx`) and an end index (`eidx`); array jobs also have increment (`incr`) values, which Moab uses to create subjobs, all executing the same script. The model for subjob creation follows the formula of end index minus start index plus increment divided by the increment value: (`eidx` - `sidx` + `incr`) / `incr`.

To illustrate, suppose an array job has a start index of 1, an end index of 100, and an increment of 1. This is an array job that creates (100 - 1 + 1) / 1 = 100 subjobs with indexes of 1, 2, 3, ..., 100. An increment of 2 produces (100 - 1 + 2) / 2 = 50 subjobs with indexes of 1, 3, 5, ..., 99. An increment of 2 with a start index of 2 produces (100 - 2 + 2) / 2 = 50 subjobs with indexes of 2, 4, 6, ..., 100. Again, subjobs are jobs in their own right that have a slightly different job naming convention `jobID[subJobIndex]` (e.g., `mycluster.45[37]` or `45[37]`).

9.10.4 Using Environment Variables to Specify Array Index Values

The script can use an environment variable to obtain the array index value to form data file and/or directory names unique to a job array's particular subjob. The following two environment variables are supplied so job scripts can recognize what index in the array they are in; use the msub command with the -V option to pass the environment parameters to the resource manager, or include the parameters in a job script; for example: `#PBS -V MOAB_JOBARRAYRANGE`.

Environment Parameter	Description
MOAB_ JOBARRAYINDEX	Used to create dataset file names, directory names, and so forth, when splitting up a single problem into multiple jobs. For example, a user may split up a problem into 20 separate jobs, each with its own input and output data files whose names contain the numbers 1-20. To illustrate, assume a user submits the 20 subjobs using two msub commands; one to submit the ten even-numbered jobs and one to submit the ten odd-numbered jobs. `msub -t job1.[1-20:2]`

Environment Parameter	Description
	`msub -t job2.[2-20:2]` The `MOAB_JOBARRAYINDEX` environment variable value populates each of the two job arrays' ten subjobs as `1`, `3`, `5`, `7`, `9`, `11`, `13`, `15`, `17` and `19` for the first job array's ten subjobs, and `2`, `4`, `6`, `8`, `10`, `12`, `14`, `16`, `18`, and `20` for the second job array's ten subjobs.
MOAB_ JOBARRAYRANGE	The count of jobs in the array.

In this topic:

9.10.4.A Control

Users can control individual subjobs in the same manner as normal jobs. In addition, a job array represents its group of subjobs and any user or administrator commands performed on a job array apply to its subjobs; for example, the command canceljob `<arrayJobId>` cancels all subjobs that belong to the job array. For more information about job control, see 3.7.20 mjobctl.

9.10.4.B Reporting

In the first example below, the parts unique to array subjobs are in bold:

```
$ checkjob -v Moab.1[1]
job Moab.1[1]

AName: Moab
State: Running
Creds:  user:user1  group:usergroup1
WallTime:   00:00:17 of 8:20:00
SubmitTime: Thu Nov  4 11:50:03
(Time Queued  Total: 00:00:00  Eligible:   INFINITY)
StartTime: Thu Nov  4 11:50:03
Total Requested Tasks: 1
Req[0]  TaskCount: 1  Partition: base
Average Utilized Procs: 0.96
NodeCount:  1
Allocated Nodes:
[node010:1]

Job Group:        Moab.1
```

```
Parent Array ID:  Moab.1
Array Index:      1
Array Range:      10
SystemID:   Moab
SystemJID:  Moab.1[1]
Task Distribution: node010
IWD:            /home/user1
UMask:          0000
Executable:     /opt/moab/spool/moab.job.3CvNjl
StartCount:     1
Partition List: base
SrcRM:          internal  DstRM: base  DstRMJID: Moab.1[1]
Flags:          ARRAYJOB,GLOBALQUEUE
StartPriority:  1
PE:             1.00
Reservation 'Moab.1[1]' (-00:00:19 -> 8:19:41  Duration: 8:20:00)
```

If the array range is not provided, the output displays all the jobs in the array:

```
$ checkjob -v Moab.1
job Moab.1

AName: Moab
Job Array Info:
  Name: Moab.1
  1 : Moab.1[1] : Running
  2 : Moab.1[2] : Running
  3 : Moab.1[3] : Running
  4 : Moab.1[4] : Running
  5 : Moab.1[5] : Running
  6 : Moab.1[6] : Running
  7 : Moab.1[7] : Running
  8 : Moab.1[8] : Running
  9 : Moab.1[9] : Running
  10 : Moab.1[10] : Running
  11 : Moab.1[11] : Running
  12 : Moab.1[12] : Running
  13 : Moab.1[13] : Running
  14 : Moab.1[14] : Running
  15 : Moab.1[15] : Running
  16 : Moab.1[16] : Running
  17 : Moab.1[17] : Running
  18 : Moab.1[18] : Running
  19 : Moab.1[19] : Running
  20 : Moab.1[20] : Running
  Totals:
    Active:   20
    Idle:     0
    Complete: 0
```

You can also use showq. This displays the array master job with a count of how many subjobs are in each queue:

```
$ showq

active jobs------------------------
JOBID              USERNAME      STATE  PROCS   REMAINING            STARTTIME

Moab.1(5)           aesplin    Running     5    00:52:41  Thu Jun 23 17:05:56
Moab.2(1)           aesplin    Running     1    00:53:41  Thu Jun 23 17:06:56
```

```
6 active jobs              6 of 6 processors in use by local jobs (100.00%)
1 of 1 nodes active      (100.00%)

eligible jobs----------------------
JOBID             USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME

Moab.2(4)          aesplin       Idle      4     1:00:00   Thu Jun 23 17:06:56

4 eligible jobs

blocked jobs-----------------------
JOBID             USERNAME      STATE  PROCS     WCLIMIT            QUEUETIME

Moab.2(1)          aesplin    Blocked      1     1:00:00   Thu Jun 23 17:06:56

1 blocked job

Total jobs:  11
```

`Moab.1` has five subjobs running. `Moab.2` has one subjob running, four waiting to run, and one that is currently blocked.

9.10.5 Job Array Cancellation Policies

Job arrays can be canceled based on the success or failure of the first subjob, the first success or failure of any subjob, or if any subjob exits with a specified exit code. The job array cancellation policies are:

Cancel Policy	Description	Exclusivity
CancelOnFirstFailure	Cancels the job array if the first subjob (`JOBARRAYINDEX = 1`) fails. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstFailure`	Mutually exclusive
CancelOnFirstSuccess	Cancels the job array if the first subjob (`JOBARRAYINDEX = 1`) succeeds. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstSuccess`	
CancelOnAnyFailure	Cancels the job array if any subjob fails. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnAnyFailure`	
CancelOnAnySuccess	Cancels the job array if any subjob succeeds. `> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnAnySuccess`	

Cancel Policy	Description	Exclusivity
CancelOnExitCode	Cancels the job array if any subjob returns the specified exit code. `> msub -t myarray[1-1000%50] -l ...,flags=CancelOnExitCode:<error code list>` The syntax for the error code list are ranges specified with a dash and individual codes delimited by a plus (+) sign, such as: 1-4+9+15 Exit codes 1-387 are accepted.	

Up to two cancellation polices can be specified for an array and the two policies must be delimited by a colon (**:**). The two 'first subjob' policies are mutually exclusive, as are the three 'any subjob' policies. You can use either 'first subjob' policy with one of the 'any subjob' policies, as shown in this example:

```
> msub -t myarray[1-1000]%50 -l ...,flags=CancelOnFirstFailure:CancelOnExitCode:3-7+11
```

9.10.6 Minimizing the Impact of Very Large Job Arrays

In this topic:

9.10.6.A Potential Problems with Very Large Job Arrays

Since Moab uses multiple files to track jobs, including each subjob of a job array, a very large job array can add thousands of files that Moab has to create and manage. On each iteration, all of those subjobs are evaluated, and those eligible to run are considered for scheduling.

The jobs also need to be migrated to the workload manager (for this discussion, we’ll just use Torque). That also adds additional work to both Moab and Torque. With default settings, it's possible that very large job arrays with a large number of job files, can cause problems in both Moab and Torque.

9.10.6.B Mitigating the Impact of Job Arrays

Fortunately, there are some settings that can be made to make Moab’s work of handling a large job array much easier. This section will help you choose the best configuration options for your site.

For the settings in this section to be most effective, users must submit large job arrays using `msub` instead of `qsub`. The reason for this is when a job array is submitted using `qsub`, Torque creates all of the subjobs in its own queues, then passes that information onto Moab during the cluster query process. This prevents Moab from managing the creation of subjobs in both Moab and Torque.

JOBMIGRATEPOLICY

This option tells Moab how to handle the migration of jobs to the resource manager, typically Torque. This option applies to both normal jobs and array subjobs. The default for this setting, if not specified, is `AUTO`, which allows Moab to decide how to migrate each job based on its current situation. If set to `IMMEDIATE`, that tells Moab to migrate all jobs to the resource manager as soon as they are created. Both of these will result in Torque having the files for all array subjobs soon after the job has been submitted, even if it does not run for days.

The best option is to configure Moab with JOBMIGRATEPOLICY JUSTINTIME. This option tells Moab to only migrate jobs once they have been scheduled to run, which minimizes the job count in the resource manager, and simplifying the management of jobs. It also simplifies cancellation of array jobs, as Torque only needs to cancel the ones it knows about, and Moab does not need to tell the resource manager about jobs it has not yet migrated.

MAXARRAYJOB

This option sets an overall limit for the total number of running, plus idle jobs that can simultaneously exist. Jobs that exceed that limit remain in the blocked state until other jobs free up space. If the available resources allow more than this number of jobs to be running, then Moab will only run that number of jobs from the array and fill in the remaining resources with other jobs. In this situation, the Idle (or 'eligible') queue will be empty.

This limit can be applied to multiple credentials, as described in 'Credentials for MAXARRAYJOB and MAXIARRAYJOB' below.

MAXIARRAYJOB

The default behavior for Moab is to migrate all of the subjobs to the Idle (or 'eligible') queue as soon as possible. Since with large job arrays it's highly unlikely that all of the subjobs can be run at once, this results in a lot of extra overhead during Moab’s scheduling. Moab is continually checking and sometimes updating the job status, and updating the job files.

By setting the option MAXIARRAYJOB, the number of subjobs being considered at once for scheduling can be controlled. The option specifies how many jobs will be simultaneously allowed in the Idle (or 'eligible') queue. The array subjobs that don't fit remain in the blocked state. As jobs migrate to the running state, Moab will migrate additional subjobs to Idle state until the limit is again reached. For example, if the setting was 200, a job array with 10,000 subjobs keeps 200 subjobs in the idle queue, runs as many as can run at once, and the remaining subjobs remain blocked until there's room for them. By selecting a value slightly greater than the maximum number of jobs that could be expected to be started at once, the huge job array should complete in the same time.

If the MAXARRAYJOB option is also configured, and Moab is able to run that limit of jobs at once, then the Idle queue will be empty no matter what this setting is.

This limit can be applied to multiple credentials, as described in 'Credentials for MAXARRAYJOB and MAXIARRAYJOB' below.

Credentials for MAXARRAYJOB and MAXIARRAYJOB

Both of these limits can be applied to USERCFG, QOSCFG, CLASSCFG, and even JOBCFG. For QOSCFG and CLASSCFG, the limit will apply cumulatively to all jobs running under that credential, which can result in some users being blocked. In the case of USERCFG, the limit is applied to the specified user, or if DEFAULT is used, it will apply the limit to each user that does not have the limit defined in their own USERCFG.

When using JOBCFG, you are defining a job template. This makes understanding the results a lot more complicated, and for this reason we recommend avoiding this unless there is a specific reason. The problem is all jobs that match a job template are aggregated under this single limit, so a higher priority large array could potentially block all other arrays from running.

The Nitro Product

Finally, if job arrays often contain serial workload (no MPI/parallel jobs) that can all run on a single node, the best solution may be the Nitro product. Instead of having a Moab job for each of these serial tasks, this product can run any number of tasks within a single Moab job. A task file is used to specify the command lines that need to be executed, and they are run with no Moab scheduling overhead. Nitro tracks the progress, and can even resume an interrupted run. A single Nitro job running trivial workload (hostname, for example) can easily run well over a thousand tasks per second.

9.10.7 Examples

Operations can be performed on individual jobs, a selection of jobs in a job array, or on the entire array.

Submitting Job Arrays

The syntax for submitting job arrays is: `msub -t [<jobname>]<indexlist> [%<limit>] arrayscript.sh`

The `<jobname>` and `<limit>` are optional. The jobname does not override the `jobID` Moab assigns to the array. When submitting an array with a jobname, Moab returns the `jobID`, which is the scheduler name followed by a unique ID.

Example 9-5:

If the scheduler name in `moab.cfg` is `Moab` (`SCHEDCFG[Moab]`), submitting an array with a jobname responds like this:

```
> msub -t myarray[1-10] job.sh

Moab.6
```

Example 9-6:

To specify that only a certain number of subjobs in the array can run at a time, use the percent sign (`%`) delimiter. In this example, only five subjobs in the array can run at a time:

```
> msub -t myarray[1-1000]%5
```

Example 9-7:

To submit a specific set of array subjobs, use the comma delimiter in the array index list:

```
> msub -t myarray[1,2,3,4]
> msub -t myarray[1-5,7,10]
```

Example 9-8:

You can use the checkjob command on either the `jobID` or the `jobname` you specified:

```
> msub -t myarray[1-2] job.sh

Moab.10

$ checkjob -v myarray
  job Moab.10

AName: myarray
Job Array Info:
   Name: Moab.10
   1 : Moab.10[1] : Running
   2 : Moab.10[2] : Running

   Sub-jobs:            2
     Active:            2 ( 100.0% )
     Eligible:          0 ( 0.0% )
     Blocked:           0 ( 0.0% )
     Completed:         0 ( 0.0% )

State: Idle
```

```
Creds:  user:tuser1  group:tgroup1
WallTime:   00:00:00 of 99:23:59:59
SubmitTime: Thu Jun  2 16:37:17
   (Time Queued  Total: 00:00:33  Eligible: 00:00:00)

Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
```

Example 9-9:

To submit a job with a step size, use a colon in the array range and specify how many jobs to step. In the example below, a step size of 2 is requested. The subjobs will be numbered according to the step size inside the index limit. The array master job name will be the same as explained above.

```
$ msub -t myarray[2-10:2] job.sh

job Moab.15

$ checkjob -v myarray #or you could use 'checkjob -v Moab.15'
job Moab.15

AName: myarray
Job Array Info:
   Name: Moab.15
   2 : Moab.15[2] : Running
   4 : Moab.15[4] : Running
   6 : Moab.15[6] : Running
   8 : Moab.15[8] : Running
   10 : Moab.15[10] : Running

   Sub-jobs:           5
     Active:           5 ( 100.0% )
     Eligible:         0 ( 0.0% )
     Blocked:          0 ( 0.0% )
     Completed:        0 ( 0.0% )

State: Idle
Creds:  user:tuser1  group:tgroup1
WallTime:   00:00:00 of 99:23:59:59
SubmitTime: Thu Jun  2 16:37:17
   (Time Queued  Total: 00:00:33  Eligible: 00:00:00)

Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
```

Related Topics

- Chapter 23: Moab Workload Manager for Grids
- 9.5 Job Dependencies

Chapter 10: General Node Administration

Moab has a very flexible and generalized definition of a node. This flexible definition, together with the fact that Moab must inter-operate with many resource managers of varying capacities, requires that Moab must possess a complete set of mechanisms for managing nodes that in some cases might be redundant with resource manager facilities.

In this chapter:

10.1 Node Attribute Types

In this section:

10.1.1 Resource Manager Specified 'Opaque' Attributes

Many resource managers support the concept of opaque node attributes, allowing you to assign arbitrary strings to a node. These strings are opaque in the sense that the resource manager passes them along to the scheduler without assigning any meaning to them. Nodes possessing these opaque attributes can then be requested by various jobs. Using certain Moab parameters, sites can assign a meaning within Moab to these opaque node attributes and extract specific node information. For example, setting the parameter

FEATUREPROCSPEEDHEADER xps causes a node with the opaque string `xps950` to be assigned a processor speed of 950 MHz within Moab.

10.1.2 Scheduler Specified Default Node Attributes

Some default node attributes can be assigned on a rack or partition basis. In addition, many node attributes can be specified globally by configuring the `DEFAULT` node template using the NODECFG parameter (i.e., `NODECFG[DEFAULT] PROCSPEED=3200`). Unless explicitly specified otherwise, nodes inherit node attributes from the associated rack or partition or from the default node template.

10.1.3 Scheduler Specified Node Attributes

The `NODECFG` parameter also enables direct per-node specification of virtually all node attributes supported via other mechanisms and also provides a number of additional attributes not found elsewhere. For example, a site admin might want to specify something such as the following:

```
NODECFG[node031] MAXJOB=2 PROCSPEED=600 PARTITION=small
```

These approaches can be mixed and matched according to the site's local needs. Precedence for the approaches generally follows the order listed earlier in cases where conflicting node configuration information is specified through one or more mechanisms.

10.2 Node Location

Nodes can be assigned three types of location information based on partitions, racks, and queues.

In this section:

10.2.1 Partitions

The first form of location assignment, the partition, allows nodes to be grouped according to physical resource constraints or policy needs. By default, jobs are not allowed to span more than one partition so partition boundaries are often valuable if an underlying network topology make certain resource allocations undesirable. Additionally, per-partition policies can be specified to grant control over how scheduling is handled on a partition by partition basis. See 6.2 Partitions for more information.

10.2.2 Racks

Rack-based location information is orthogonal to the partition based configuration and is mainly an organizational construct. In general rack based location usage, a node is assigned both a rack and a slot number. This approach has descended from the IBM SP2 organizational approach where a rack can contain any number of slots, but typically contains between 1 and 99. Using the rack and slot number combo, individual compute nodes can be grouped and displayed in a more ordered manner in certain Moab commands (i.e., showstate). Currently, rack information can only be specified directly by the system via the SDR interface on SP2/Loadleveler systems. In all other systems, this information must be specified using an information service or specified manually using the RACK, SLOT, and SIZE attributes of the NODECFG parameter.

Sites can arbitrarily assign nodes to racks and rack slots without impacting scheduling behavior. Neither rack numbers nor rack slot numbers need to be contiguous; their use is simply for convenience purposes in displaying and analyzing compute resources.

Example 10-1: rack and slot

When specifying node and rack information, slot values must be in the range of 1 to 99, and racks must be in the range of 1 to 399:

```
NODECFG[node024] RACK=1 SLOT=1
NODECFG[node025] RACK=1 SLOT=2
NODECFG[node026] RACK=2 SLOT=1 PARTITION=special
...
```

10.2.3 Queues

Some resource managers allow queues (or classes) to be defined and then associated with a subset of available compute resources. With systems such as Loadleveler or PBSPro these queue to node mappings are automatically detected. On resource managers that do not provide this service, Moab provides alternative mechanisms for enabling this feature.

Torque Queue to Node Mapping

Under Torque, queue to node mapping can be accomplished by using the qmgr command to set the queue acl_hosts parameter to the mapping hostlist desired. Further, the `acl_host_enable` parameter should be set to `False`.

> Setting `acl_hosts` and then setting `acl_host_enable` to `True` constrains the list of hosts from which jobs can be submitted to the queue.

Example 10-2:

The following example highlights this process and maps the queue `debug` to the nodes `host14` through `host17`:

```
> qmgr
Max open servers: 4
Qmgr: set queue debug acl_hosts = "host14,host15,host16,host17"
Qmgr: set queue debug acl_host_enable = false
Qmgr: quit
```

> All queues that do not have `acl_hosts` specified are global; that is, they show up on every node. To constrain these queues to a subset of nodes, each queue requires its own `acl_hosts` parameter setting.

10.2.4 Node Selection

When selecting or specifying nodes either via command line tools or via configuration file based lists, Moab offers three types of node expressions that can be based on node lists, exact lists, node ranges, or regular expressions.

In this topic:

10.2.4.A Node Lists

Node lists can be specified as one or more comma or whitespace delimited node IDs. Specified node IDs can be based on either short or fully qualified hostnames. Each element will be interpreted as a regular expression.

```
SRCFG[basic]  HOSTLIST=cl37.icluster,ax45,ax46
...
```

10.2.4.B Exact Lists

When Moab receives a list of nodes it will, by default, interpret each element as a regular expression. To disable this and have each element interpreted as a string node name, the `l:` can be used as in the following example:

```
> setres l:n00,n01,n02
```

10.2.4.C Node Range

Node lists can be specified as one or more comma or whitespace delimited node ranges. Each node range can be based using either `<STARTINDEX>-<ENDINDEX>` or `<HEADER>[<STARTINDEX>-<ENDINDEX>]` format. To explicitly request a range, the node expression must be preceded with the string `r:` as in the following example:

```
> setres r:37-472,513,516-855
```

When you specify a `<HEADER>` for the range, note that it must only contain alphabetical characters. As always, the range must be numeric.

```
CLASSCFG[long] HOSTLIST=r:anc-b[37-472]
```

Only one expression is allowed with node ranges.

By default, Moab attempts to extract a node's node index assuming this information is built into the node's naming convention. If needed, this information can be explicitly specified in the Moab configuration file using NODECFG's `NODEINDEX` attribute, or it can be extracted from alternatively formatted node IDs by specifying the NODEIDFORMAT parameter.

10.2.4.D Node Regular Expression

Node lists can also be specified as one or more comma or whitespace delimited regular expressions. Each node regular expression must be specified in a format acceptable by the standard C regular expression libraries that allow support for wildcard and other special characters such as the following:

`*` (asterisk)

`.` (period)

`[ ]` (left and right bracket)

`^` (caret)

`$` (dollar)

Node lists are by default interpreted as a regular expression but can also be explicitly requested with the string `x:` as in the following examples:

```
# select nodes cl30 thru cl55
SRCFG[basic]  HOSTLIST=x:cl[34],cl5[0-5]
...
```

```
# select nodes cl30 thru cl55
SRCFG[basic]  HOSTLIST=cl[34],cl5[0-5]
...
```

To control node selection search ordering, set the `OBJECTELIST` parameter to one of the following options: exact, range, regex, rangere, or rerange.

10.3 Node Attributes

In this section:

10.3.1 Configurable Node Attributes

Nodes can possess a large number of attributes describing their configuration, which are specified using the NODECFG parameter. The majority of these attributes such as operating system or configured network interfaces can only be specified by the direct resource manager interface. However, the number and detail of node attributes varies widely from resource manager to resource manager. Sites often have interest in making scheduling decisions based on scheduling attributes not directly supplied by the resource manager. Configurable node attributes are listed in the following table:

Node Attributes

ACCESS	NODEAVAILABILITYPOLICY	PROVRM
ARCH	NODEINDEX	RACK
CHARGERATE	NODETYPE	RADISK
COMMENT	OS	RAMEM
ENABLEPROFILING	OSLIST	RCDISK
FEATURES	PARTITION	RCMEM
GRES	POWERPOLICY	RCPROC
MAXJOB	PREEMPTMAXCPULOAD	RCSWAP
MAXJOBPERUSER	PREEMPTMINMEMAVAIL	SIZE
MAXLOAD	PREEMPTPOLICY	SLOT
MAXPE	PRIORITY	SPEED
MAXPEPERJOB	PRIORITYF	TRIGGER
MAXPROC	PROCSPEED	VARIABLE

Attribute	Description
ACCESS	The node access policy, which can be one of SHARED, SHAREDONLY, SINGLEJOB, SINGLETASK, or SINGLEUSER. See 10.4.2 Node Access Policies for more information. `NODECFG[node013] ACCESS=singlejob`
ARCH	The node's processor architecture. `NODECFG[node013] ARCH=opteron`

CHARGERATE	Enables you to assign specific charging rates to the usage of particular resources. The CHARGERATE value can be specified as a floating point value and is integrated into a job's total charge (as documented in 5.5 Accounting, Charging, and Allocation Management).
	`NODECFG[DEFAULT] CHARGERATE=1.0`
	`NODECFG[node003] CHARGERATE=1.5`
	`NODECFG[node022] CHARGERATE=2.5`

COMMENT	Enables you to annotate a node via the configuration file to indicate special information regarding this node to both users and admins. The COMMENT value can be specified as a quote delimited string as shown in the example below. Comment information is visible using checknode, mdiag, and Moab Cluster Manager. `NODECFG[node013] COMMENT="Login Node"`

Attribute	Description
ENABLEPROFILING	Enables you to track node state over time. This information is available using showstats -n. `NODECFG[DEFAULT] ENABLEPROFILING=TRUE`
FEATURES	Not all resource managers allow specification of opaque node features (also known as node properties). For these systems, the NODECFG parameter can be used to directly assign a list of node features to individual nodes. To append node features, use `FEATURES=<X>`; to overwrite or remove a node's features, you must update them in your Moab configuration file or resource manager. `NODECFG[node013] FEATURES=gpfs,fastio` *Node `node013` now has features `gpfs` and `fastio` in addition to any other features configured in this file or the resource manager.* The total number of supported node features is limited as described in MMAX_ATTR. If supported by the resource manager, the resource manager specific manner of requesting node features/properties within a job can be used. (Within Torque, use `qsub -l nodes=<NODECOUNT>:<NODEFEATURE>`.) However, if either not supported within the resource manager or if support is limited, the Moab feature resource manager extension can be used.
GRES	Many resource managers do not allow specification of consumable generic node resources. For these systems, the NODECFG parameter can be used to directly assign a list of consumable generic attributes to individual nodes or to the special pseudo-node global, which provides shared cluster (floating) consumable resources. To set/overwrite a node's generic resources, use `GRES=<NAME>[:<COUNT>]`. See 10.7 Managing Consumable Generic Resources. `NODECFG[node013] GRES=quickcalc:20`
MAXJOB	See 10.4.1.A MAXJOB for details.

Attribute	Description
MAXJOBPERUSER	See 10.4.1.B MAXJOBPERUSER for details.
MAXLOAD	See 10.4.1.D MAXLOAD for details.
MAXPE	See 10.4.1.E MAXPE for details.

MAXPEPERJOB	Maximum allowed Processor Equivalent per job on this node. A job will not be allowed to run on this node if its PE exceeds this number.
	`NODECFG[node024] MAXPEPERJOB=10000`
	`...`

MAXPROC	Maximum dedicated processors allowed on this node. No jobs are scheduled on this node when this number is reached. See 10.4 Node Specific Policies for more information.
	`NODECFG[node024] MAXPROC=8`
	`...`

NODEAVAILABILITYPOLICY	Specifies how available node resources are reported.
	This sets the NODEAVAILABILITYPOLICY at the local level and uses a different format from the NODEAVAILIBILITYPOLICY server parameter. See the parameter NODEAVAILABILITYPOLICY.
	`NODECFG[node00]`
	`NODEAVAILABILITYPOLICY=DEDICATED:PROC,UTILIZED:MEM,COMBINED:DISK`

NODEINDEX	The node's index. See 10.2 Node Location for details.
NODETYPE	The `NODETYPE` attribute is most commonly used in conjunction with an accounting manager such as Moab Accounting Manager. In these cases, each node is assigned a node type and within the accounting manager, each node type is assigned a charge rate. For example, a site admin might want to charge users more for using large memory nodes and might assign a node type of `BIGMEM` to these nodes. The accounting manager will then charge a premium rate for jobs using `BIGMEM` nodes. See 5.5 Accounting, Charging, and Allocation Management for more information. Node types are specified as simple strings. If no node type is

Attribute	Description
	explicitly set, the node will possess the default node type of `DEFAULT`. Node type information can be specified directly using `NODECFG` or through use of the FEATURENODETYPEHEADER parameter. `NODECFG[node024] NODETYPE=BIGMEM`
OS	This attribute specifies the node's operating system. `NODECFG[node013] OS=suse15` Because the Torque operating system overwrites the Moab operating system, change the operating system with opsys instead of **OS** if you are using Torque, and/or you can use RMCFG[torque] FLAGS=IgnOS to override that.
OSLIST	This attribute specifies the list of operating systems the node can run. `NODECFG[compute002] OSLIST=linux,windows`
PARTITION	See 10.2.1 Partitions for details.
POWERPOLICY	The `POWERPOLICY` can be set to `OnDemand` or `STATIC`. It defaults to `STATIC` if not set. If set to `STATIC`, Moab will never automatically change the power status of a node. If set to `OnDemand`, Moab will turn the machine off and on based on workload and global settings. See Chapter 15: Green Computing for more information.

PREEMPTMAXCPULOAD	If the node CPU load exceeds the specified value, any batch jobs running on the node are preempted using the preemption policy specified with the node's PREEMPTPOLICY attribute. If this attribute is not specified, the global default policy specified with PREEMPTPOLICY parameter is used. See the algorithm PRIORITY for more information.
	`NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"`
	`NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMAXCPULOAD=1.2`
	`...`

PREEMPTMINMEMAVAIL	If the available node memory drops below the specified value, any batch jobs running on the node are preempted using the

Attribute	Description

	preemption policy specified with the node's PREEMPTPOLICY attribute. If this attribute is not specified, the global default policy specified with PREEMPTPOLICY parameter is used. See the algorithm PRIORITY for more information.
	`NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"`
	`NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMINMEMAVAIL=256`
	`...`

PREEMPTPOLICY	If any node preemption policies are triggered (such as PREEMPTMAXCPULOAD or PREEMPTMINMEMAVAIL) any batch jobs running on the node are preempted using this preemption policy if specified. If not specified, the global default preemption policy specified with PREEMPTPOLICY parameter is used. See the algorithm PRIORITY for more information.
	`NODECFG[node024] PRIORITY=-150 COMMENT="NFS Server Node"`
	`NODECFG[node024] PREEMPTPOLICY=CANCEL PREEMPTMAXCPULOAD=1.2`
	`...`

PRIORITY	The `PRIORITY` attribute specifies the fixed node priority relative to other nodes. It is only used if the parameter NODEALLOCATIONPOLICY is set to `PRIORITY`. The default node priority is 0. A default cluster-wide node priority can be set by configuring the `PRIORITY` attribute of the `DEFAULT` node. See the algorithm PRIORITY for more information.
	`NODEALLOCATIONPOLICY  PRIORITY`
	`NODECFG[node024] PRIORITY=120`
	`...`

PRIORITYF	The `PRIORITYF` attribute specifies the function to use when calculating a node's allocation priority specific to a particular job. It is only used if the parameter NODEALLOCATIONPOLICY is set to `PRIORITY`. The default node priority function sets a node's priority exactly equal to the configured node priority. The priority function enables you to indicate that various environmental considerations such as node load, reservation affinity, and ownership be taken into account also using the following format: `<COEFFICIENT> * <ATTRIBUTE> [ + <COEFFICIENT> * <ATTRIBUTE> ]...` `<ATTRIBUTE>` is an entry from the list of PRIORITY components in the Allocation Algorithms table. A default cluster-wide node priority function can be set by configuring the `PRIORITYF` attribute of the `DEFAULT` node.

Attribute	Description

	See the algorithm PRIORITY for more information.
	`NODEALLOCATIONPOLICY  PRIORITY`
	`NODECFG[node024] PRIORITYF='APROC + .01 * AMEM - 10 * JOBCOUNT'`
	`...`

PROCSPEED	Knowing a node's processor speed can help the scheduler improve intra-job efficiencies by allocating nodes of similar speeds together. This helps reduce losses due to poor internal job load balancing. Moab's node set scheduling policies allow you to control processor speed based allocation behavior. Processor speed information is specified in MHz and can be indicated directly using NODECFG or through use of the FEATUREPROCSPEEDHEADER parameter.

PROVRM	Provisioning resource managers can be specified on a per node basis. This enables flexibility in mixed environments. If the node does not have a provisioning resource manager, the default provisioning resource manager will be used. The default is always the first one listed in `moab.cfg`.
	`RMCFG[prov] TYPE=NATIVE RESOURCETYPE=PROV`
	`RMCFG[prov] PROVDURATION=10:00`
	`RMCFG[prov] NODEMODIFYURL=exec://$HOME/tools/os.switch.pl`
	`...`
	`NODECFG[node024] PROVRM=prov`

RACK	The rack associated with the node's physical location. Values range from 1 to 400. See 10.2 Node Location for details.
RADISK	Jobs can request a certain amount of disk space through the RM Extension String's `DDISK` parameter. When done this way, Moab can track the amount of disk space available for other jobs. `RADISK` is used as an initial value and is subsequently determined by `RCDISK - <JOB USAGE>`.
RAMEM	The total available memory on a node. Jobs can request a certain amount of real memory (RAM) in MB through the RM Extension String's `DMEM` parameter. When done this way, Moab can track the amount of memory available for other jobs. `RAMEM` is used as an initial value and is subsequently determined by `RCMEM - <JOB USAGE>`.
RCDISK	Jobs can request a certain amount of disk space (in MB) through

Attribute	Description
	the RM Extension String's `DDISK` parameter. When done this way, Moab can track the amount of disk space available for other jobs. The `RCDISK` attribute constrains the amount of disk reported by a resource manager while the `RADISK` attribute specifies the amount of disk available to jobs. If the resource manager does not report available disk, the `RADISK` attribute should be used.

RCMEM	Jobs can request a certain amount of real memory (RAM) in MB through the RM Extension String's `DMEM` parameter. When done this way, Moab can track the amount of memory available for other jobs. The `RCMEM` attribute constrains the amount of RAM reported by a resource manager while the `RAMEM` attribute specifies the amount of RAM available to jobs. If the resource manager does not report available memory, the `RAMEM` attribute should be used.
	
	Note that memory reported by the resource manager will override the configured value unless a trailing caret (`^`) is used.
	
	`NODECFG[node024] RCMEM=2048`
	`...`
	
	If the resource manager does not report any memory, then Moab will assign `node024` `2048` MB of memory.
	
	`NODECFG[node024] RCMEM=2048^`
	`...`
	
	Moab will assign `2048` MB of memory to `node024` regardless of what the resource manager reports.

RCPROC	The `RCPROC` specifies the number of processors available on a compute node.
	
	`NODECFG[node024] RCPROC=8`
	`...`

RCSWAP	Jobs can request a certain amount of swap space in MB. `RCSWAP` works similarly to `RCMEM`. Setting `RCSWAP` on a node will set the swap but can be overridden by swap reported by the resource manager. If the trailing caret (`^`) is used, Moab will ignore the swap reported by the resource manager and use the configured amount.

Attribute	Description

	```
	NODECFG[node024] RCSWAP=2048
	...
	```
	 If the resource manager does not report any memory, Moab will assign `node024` `2048` MB of swap.
	```
	NODECFG[node024] RCSWAP=2048^
	...
	```
	 Moab will assign `2048` MB of swap to `node024` regardless of what the resource manager reports.

SIZE	The number of slots or size units consumed by the node. This value is used in graphically representing the cluster using the command showstate or Moab Cluster Manager. See 10.2 Node Location for details. For display purposes, valid size values include `1`, `2`, `3`, `4`, `6`, `8`, `12`, and `16`.
	```
	NODECFG[node024] SIZE=2
	...
	```

SLOT	The first slot in the rack associated with the node's physical location. Values range from 1 to `MMAX_RACKSIZE` (default=`64`). See 10.2 Node Location for details.
SPEED	Because today's processors have multiple cores and adjustable clock frequency, this feature has no meaning and is deprecated. The `SPEED` specification must be in the range of 0.01 to 100.0.
TRIGGER	See 17.1 About Object Triggers for information.

VARIABLE	Variables associated with the given node, which can be used in job scheduling. See the PREF job extension.
	```
	NODECFG[node024] VARIABLE=var1
	...
	```

10.3.2 Node Features/Node Properties

A node feature (or node property) is an opaque string label that is associated with a compute node. Each compute node can have any number of node features assigned to it, and jobs can request allocation of nodes that have specific features assigned. Node features are labels and their association with a compute node is not conditional, meaning they cannot be consumed or exhausted.

Node features can be assigned by the resource manager, and this information can be imported by Moab or node features can be specified within Moab directly. Moab supports hyphens and underscores in node feature names.

As a convenience feature, certain node attributes can be specified via node features using the parameters listed in the following table:

Parameter	Description
FEATURENODETYPEHEADER	Set Node Type
FEATUREPARTITIONHEADER	Set Partition
FEATUREPROCSPEEDHEADER	Set Processor Speed
FEATURERACKHEADER	Set Rack
FEATURESLOTHEADER	Set Slot

Example 10-3:

```
FEATUREPARTITIONHEADER  par
FEATUREPROCSPEEDHEADER  cpu
```

Related Topics

- 11.3 Resource Manager Extensions - PREF
- Specifying Node Features (Node Properties) in the *Torque Resource Manager Administrator Guide*
- Appendix A: Moab Parameters - NODECFG (configuring node features in Moab)
- 3.7.17 mdiag -t (viewing feature availability breakdown)
- 10.7.1 Differences Between Node Features and Consumable Resources

10.4 Node Specific Policies

Node policies within Moab allow specification of not only how the node's load should be managed, but who can use the node, and how the node and jobs should respond to various events. These policies allow a site admin to specify on a node by node basis what the node will and will not support. Node policies can be applied to specific nodes or applied system-wide using the specification `NODECFG[DEFAULT] ...`.

In this section:

10.4.1 Node Usage/Throttling Policies

In this topic:

10.4.1.A MAXJOB

This policy constrains the number of total independent jobs a given node can run simultaneously. It can only be specified via the NODECFG parameter.

10.4.1.B MAXJOBPERUSER

Constrains the number of total independent jobs a given node can run simultaneously associated with any single user. It can only be specified via the NODECFG parameter.

10.4.1.C MAXJOBPERGROUP

Constrains the number of total independent jobs a given node can run simultaneously associated with any single group. It can only be specified via the NODECFG parameter. Setting MAXLOAD here to -1 unsets the NODEMAXLOAD parameter setting.

10.4.1.D MAXLOAD

Constrains the CPU load the node will support as opposed to the number of jobs. This maximum load policy can also be applied system wide using the parameter NODEMAXLOAD.

10.4.1.E MAXPE

Constrains the number of total dedicated processor-equivalents a given node can support simultaneously. It can only be specified via the NODECFG parameter. For more information, see the subsection 'PE (Processor Equivalent) Calculation' under 2.3 Scheduling Environment.

10.4.1.F MAXPROC

Constrains the number of total dedicated processors a given node can support simultaneously. It can only be specified via the NODECFG parameter.

10.4.1.G MAXPROCPERUSER

Constrains the number of total processors a given node can have dedicated to any single user. It can only be specified via the NODECFG parameter.

10.4.1.H MAXPROCPERGROUP

Constrains the number of total processors a given node can have dedicated to any single group. It can only be specified via the NODECFG parameter.

Node throttling policies are used strictly as constraints. If a node is defined as having a single processor or the parameter NODEACCESSPOLICY is set to SINGLETASK, and a MAXPROC policy of 4 is specified, Moab will not run more than one task per node. A node's configured processors must be specified so that multiple jobs can run and then the MAXJOB policy will be effective. The number of configured processors per node is specified on a resource manager specific basis. PBS, for example, allows this to be adjusted by setting the number of virtual processors with the np parameter for each node in the PBS nodes file.

Example 10-4: NODECFG

```
NODECFG[node024] MAXJOB=4 MAXJOBPERUSER=2
NODECFG[node025] MAXJOB=2
NODECFG[node026] MAXJOBPERUSER=1
```

```
NODECFG[DEFAULT] MAXLOAD=2.5
...
```

10.4.2 Node Access Policies

While most sites require only a single cluster wide node access policy (commonly set using the parameter NODEACCESSPOLICY), it is possible to specify this policy on a node by node basis using the `ACCESS` attributes of the NODECFG parameter. This attribute can be set to any of the node access policy values listed in the section Node Access Policies.

Example 10-5: ACCESS

To set a global policy of `SINGLETASK` on all nodes except nodes 13 and 14, use the following:

```
# by default, enforce dedicated node access on all nodes
NODEACCESSPOLICY  SINGLETASK
# allow nodes 13 and 14 to be shared
NODECFG[node13]   ACCESS=SHARED
NODECFG[node14]   ACCESS=SHARED
```

Related Topics

- 3.7.21 mnodectl

10.5 Managing Shared Cluster Resources (Floating Resources)

This section describes how to configure, request, and reserve cluster file system space and bandwidth, software licenses, and generic cluster resources.

In this section:

10.5.1 Shared Cluster Resource

Shared cluster resources such as file systems, networks, and licenses can be managed through creating a pseudo-node. You can configure a pseudo-node via the NODECFG parameter much as a normal node would be but additional information is required to allow the scheduler to contact and synchronize state with the resource.

In the following example, a license manager is added as a cluster resource by defining the GLOBAL pseudo-node and specifying how the scheduler should query and modify its state:

```
NODECFG[GLOBAL] RMLIST=NATIVE
NODECFG[GLOBAL] QUERYCMD=/usr/local/bin/flquery.sh
NODECFG[GLOBAL] MODIFYCMD=/usr/local/bin/flmodify.sh
```

In some cases, pseudo-node resources might be very comparable to node-locked generic resources however there are a few fundamental differences that determine when one method of describing resources should be used over the other. The following table contrasts the two resource types:

Attribute	Pseudo-Node	Generic Resource
Node-Locked	No - Resources can be encapsulated as an independent node.	Yes - Must be associated with an existing compute node.
Requires exclusive batch system control over resource	No - Resources (such as file systems and licenses) can be consumed both inside and outside of batch system workload.	Yes - Resources must only be consumed by batch workload. Use outside of batch control results in loss of resource synchronization.
Allows scheduler level allocation of resources	Yes - If required, the scheduler can take external administrative action to allocate the resource to the job.	No - The scheduler can only maintain logical allocation information and cannot take any external action to allocate resources to the job.

10.5.2 Configuring Generic Consumable Floating Resources

Consumable floating resources are configured in the same way as node-locked generic resources with the exception of using the GLOBAL node instead of a particular node:

```
NODECFG[GLOBAL] GRES=tape:4,matlab:2
...
```

In this setup, four resources of type tape and 2 of type matlab are floating and available across all nodes.

Requesting Consumable Floating Resources

Floating resources are requested on a per task basis using Native Resource Manager job submission methods or using the GRES resource manager extensions.

10.5.3 Configuring Cluster File Systems

Moab allows both the file space and bandwidth attributes or a cluster file system to be tracked, reserved, and scheduled. With this capability, a job or reservation can request a particular quantity of file space and a required amount of I/O bandwidth to this file system. While file system resources are managed as a cluster generic resource, they are specified using the FS attribute of the NODECFG parameter as in the following example:

```
NODECFG[GLOBAL] FS=PV1:10000@100,PV2:5000@100
...
```

In this example, PV1 defines a 10 GB file system with a maximum throughput of 100 MB/s while PV2 defines a 5 GB file system also possessing a maximum throughput of 100 MB/s.

A job can request cluster file system resources using the fs resource manager extension. For a Torque based system, the following could be used:

```
> qsub -l nodes=1,walltime=1:00:00 -W x=fs:10@50
```

10.5.4 Configuring Cluster Licenses

Jobs can request and reserve software licenses using native methods or using the GRES resource manager extension. If the cluster license manager does not support a query interface, license availability can be specified within Moab using the GRES attribute of the NODECFG parameter.

Example 10-6: Configure Moab to support four floating quickcalc and two floating matlab licenses.

```
NODECFG[GLOBAL] GRES=quickcalc:4,matlab:2
...
```

Example 10-7: Submit a Torque job requesting a node-locked or floating quickcalc license.

```
> qsub -l nodes=1,software=quickcalc,walltime=72000 testjob.cmd
```

10.5.5 Configuring Generic Resources as Features

Moab can be configured to treat generic resources as features in order to provide more control over server access. For instance, if a node is configured with a certain GRES and that GRES is turned off, jobs requesting the node will not run. To turn a GRES into a feature, set the FEATUREGRES attribute of GRESCFG to TRUE in the moab.cfg file:

```
GRESCFG[gres1] FEATUREGRES=TRUE
```

Moab now treats `gres1` as a scheduler-wide feature rather than a normal generic resource.

Note that jobs are submitted normally using the same GRES syntax.

> If you are running a grid, verify that `FEATUREGRES=TRUE` is set on all members of the grid.

> You can safely upgrade an existing cluster to use the feature while jobs are running. If you are in a grid, upgrade all clusters at the same time.

Two methods exist for managing GRES features: via Moab commands and via the resource manager. Using Moab commands means that feature changes are not checkpointed; they do not remain in place when Moab restarts. Using the resource manager causes changes to be reported by the resource manager, so any changes made before a Moab restart are still present after it.

These methods are mutually exclusive. Use one or the other, but do not mix methods.

In this topic:

10.5.5.A Managing Feature GRES via Moab Commands

In the following example, `gres1` and `gres2` are configured in the `moab.cfg` file. `gres1` is not currently functioning correctly, so it is set to 0, turning the feature off. Values above 0 and non-specified values turn the feature on.

```
NODECFG[GLOBAL] GRES=gres1:0
NODECFG[GLOBAL] GRES=gres2:10000
GRESCFG[gres1] FEATUREGRES=TRUE
GRESCFG[gres2] FEATUREGRES=TRUE
```

Moab now treats `gres1` and `gres2` as features.

To verify that this is set up correctly, run mdiag -S -v. It returns the following:

```
> mdiag -S -v
...
  Scheduler FeatureGres: gres1:off,gres2:on
```

Once Moab has started, use mschedctl -m to modify whether the feature is turned on or off:

```
mschedctl -m sched featuregres:gres1=on

INFO: FeatureGres 'gres1' turned on
```

You can verify that the feature turned on or off by once again running *mdiag -S -v*.

If Moab restarts, it will not checkpoint the state of these changed feature generic resources. Instead, it will read the `moab.cfg` file to determine whether the feature GRES is on or off.

With feature GRES configured, jobs are submitted normally, requesting GRES type `gres1` and `gres2`. Moab ignores GRES counts and reads the feature simply as on or off.

```
> msub -l nodes=1,walltime=600,gres=gres1

1012
> checkjob 1012
job 1012

AName: STDIN
State: Running
.....
StartTime: Tue Jul 3 15:33:28
Feature GRes: gres1
Total Requested Tasks: 1
```

If you request a feature that is currently turned off, the state is not reported as `Running`, but as `Idle`. A message such as the following returns:

```
BLOCK MSG: requested feature gres 'gres2' is off
```

10.5.5.B Managing Feature GRES via the Resource Manager

You can automate the process of having a feature GRES turn on and off by setting up an external tool and configuring Moab to query the tool the same way that Moab queries a license manager. For example:

```
RMCFG[myRM] CLUSTERQUERYURL=file:///$HOME/tools/myRM.dat TYPE=NATIVE
RESOURCETYPE=LICENSE

GRESCFG[gres1] FEATUREGRES=TRUE
GRESCFG[gres2] FEATUREGRES=TRUE
```

`LICENSE` means that the resource manager does not contain any compute resources and that Moab should not attempt to use it to manage any jobs (start, cancel, submit, etc.).

The `myRM.dat` file should contain something such as the following:

```
GLOBAL state=Idle cres=gres1:0,gres2:10
```

External tools can easily update the file based on filesystem availability. Switching any of the feature GRES to 0 turns it off and switching it to a positive value turns it on. If you use this external mechanism, you do not need to use *mschedctl -m* to turn a feature GRES on or off. You also do not need to worry about whether Moab has checkpointed the information or not, since the information is provided by the resource manager and not by any external commands.

10.5.6 Configuring Generic Resources as Licenses

Moab can be configured to treat generic resources as licenses in order to distinguish them as licenses in terms of tracking and charging with the accounting manager. To turn a GRES into a license, set the LICENSE attribute of GRESCFG to TRUE in the moab.cfg file. For example:

```
GRESCFG[matlab] LICENSE=TRUE
```

Moab will pass the matlab generic resource to the accounting manager in the Licenses property rather than the Resources property.

Related Topics

- 11.5 Managing Resources Directly with the Native Resource Manager Interface

10.6 Managing Node State

There are multiple models where Moab can operate allowing it to either honor the node state set by an external service or locally determine and set the node state. This section covers the following:

- Identifying meanings of particular node states
- Specifying node states within locally developed services and resource managers
- Adjusting node state within Moab based on load, policies, and events

In this section:

10.6.1 Node State Definitions

State	Definition
Down	Node is either not reporting status, is reporting status but failures are detected, or is reporting status but has been marked down by an admin.
Idle	Node is reporting status, currently is not executing any workload, and is ready to accept additional workload.
Busy	Node is reporting status, currently is executing workload, and cannot accept additional workload due to load.
Running	Node is reporting status, currently is executing workload, and can accept additional workload.
Drained	Node is reporting status, currently is not executing workload, and cannot accept additional workload due to administrative action.
Draining	Node is reporting status, currently is executing workload, and cannot accept additional workload due to administrative action.

10.6.2 Specifying Node States within Native Resource Managers

Native Resource Managers can report node state implicitly and explicitly, using `NODESTATE`, `LOAD`, and other attributes. See 11.5 Managing Resources Directly with the Native Resource Manager Interface for more information.

10.6.3 Moab Based Node State Adjustment

Node state can be adjusted based on reported processor, memory, or other load factors. It can also be adjusted based on reports of one or more resource managers in a multi-resource manager configuration. Also, both generic events and generic metrics can be used to adjust node state.

Torque health scripts (allow compute nodes to detect and report site specific failures).

10.6.4 Adjusting Scheduling Behavior Based on Reported Node State

Based on reported node state, Moab can support various policies to make better use of available resources. For more information, see Chapter 15: Green Computing.

Down State

- JOBACTIONONNODEFAILURE parameter (cancel/requeue jobs if allocated nodes fail).
- Triggers (take specified action if failure is detected).

10.6.5 Adding or Removing Nodes

When a node has been deleted by a resource manager and the resource manager no longer reports data for the node, the node continues to exist in Moab until the next restart.

As a best practice, we recommend adding or removing nodes only during cluster maintenance, rather than during periods of production activity. A restart of Moab must follow the addition and/or removal of nodes. This guarantees that Moab will handle nodes in a reliable, predictable way. If you want to remove nodes from service, but cannot immediately restart Moab after doing so, we recommend marking the nodes offline (for example, with `pbsnodes -o <nodeID>` or `mnodectl -m state=down <nodeID>`) and/or placing an administrative reservation over the nodes, until such time as you can follow the recommended removal procedure during a planned maintenance window.

Related Topics

- 11.5 Managing Resources Directly with the Native Resource Manager Interface
- 11.7 License Management
- 4.4 Node Availability Policies
- NODEMAXLOAD parameter
- Chapter 15: Green Computing

10.7 Managing Consumable Generic Resources

Each time a job is allocated to a compute node, it consumes one or more types of resources. Standard resources such as CPU, memory, disk, network adapter bandwidth, and swap are automatically tracked and consumed by Moab. However, in many cases, additional resources may be provided by nodes and consumed by jobs that must be tracked. The purpose of this tracking may include accounting, billing, or the prevention of resource over-subscription. Generic consumable resources can be used to manage software licenses, I/O usage, bandwidth, application connections, or any other aspect of the larger compute environment; they can be associated with compute nodes, networks, storage systems, or other real or virtual resources.

These additional resources can be managed within Moab by defining one or more generic resources. The first step in defining a generic resource involves naming the resource. Generic resource availability can then be associated with various compute nodes and generic resource usage requirements can be associated with jobs.

In this section:

10.7.1 Differences Between Node Features and Consumable Resources

A node feature (or node property) is an opaque string label that is associated with a compute node. Each compute node can have any number of node features assigned to it and jobs can request allocation of nodes that have specific features assigned. Node features are labels and their association with a compute node is not conditional, meaning they cannot be consumed or exhausted.

10.7.2 Configuring Node-locked Consumable Generic Resources

Consumable generic resources are supported within Moab using either direct configuration or resource manager auto-detect (as when using Torque and accelerator hardware). For direct configuration, node-locked consumable generic resources (or generic resources) are specified using the `NODECFG` parameter's `GRES` attribute. This attribute is specified using the format `<ATTR>:<COUNT>` as in the following example:

```
NODECFG[titan001] GRES=tape:4
NODECFG[login32]  GRES=matlab:2,prime:4
NODECFG[login33]  GRES=matlab:2
...
```

By default, Moab supports up to 128 independent generic resource types.

In this topic:

10.7.2.A Requesting Consumable Generic Resources

Generic resources can be requested on a per task or per job basis using the `GRES` resource manager extension. If the generic resource is located on a compute node, requests are by default interpreted as a per task request. If the generic resource is located on a shared, cluster-level resource (such as a network or storage system), then the request defaults to a per job interpretation.

Generic resources are specified per task, not per node. When you submit a job, each processor becomes a task. For example, a job asking for `nodes=3:ppn=4,gres=test:5` asks for 60 gres of type test ((3*4 processors)*5).

If using Torque, the GRES or software resource can be requested as in the following examples:

Example 10-8: Per Task Requests

Moab determines that compute nodes exist that possess the requested generic resource:

```
NODECFG[compute001] GRES=dvd:2 SPEED=2200
NODECFG[compute002] GRES=dvd:2 SPEED=2200
NODECFG[compute003] GRES=dvd:2 SPEED=2200
NODECFG[compute004] GRES=dvd:2 SPEED=2200
NODECFG[compute005] SPEED=2200
NODECFG[compute006] SPEED=2200
NODECFG[compute007] SPEED=2200
NODECFG[compute008] SPEED=2200
```

```
# submit job which will allocate only from nodes 1 through 4 requesting one dvd per
task
> qsub -l nodes=2,walltime=100,gres=dvd job.cmd
```

A compute node is a node object that possesses processors on which compute jobs actually execute. License server, network, and storage resources are typically represented by non-compute nodes. Because compute nodes exist with the requested generic resource, Moab interprets this job as requesting two compute nodes each of which must also possess a `DVD` generic resource.

Example 10-9: Per Job Requests

Moab determines that there exist no compute nodes that also possess the generic resource `bandwidth` so this job is translated into a multiple-requirement—multi-req—job:

```
NODECFG[network] PARTITION=shared GRES=bandwidth:2000000
```

```
# submit job which will allocate 2 nodes and 10000 units of network bandwidth
> qsub -l nodes=2,walltime=100,gres=bandwidth:10000 job.cmd
```

Moab creates a job that has a requirement for two compute nodes and a second requirement for `10000bandwidth` generic resources. Because this is a multi-req job, Moab knows that it can locate these needed resources separately.

10.7.2.B Using Generic Resource Requests in Conjunction with other Constraints

Jobs can explicitly specify generic resource constraints. However, if a job also specifies a hostlist, the hostlist constraint overrides the generic resource constraint if the request is for per task allocation. In the Per Task Requests example, if the job also specified a hostlist, the `DVD` request is ignored.

10.7.2.C Requesting Resources with No Generic Resources

In some cases, it is valuable to allocate nodes that currently have no generic resources available. This can be done using the special value `none` as in the following example:

```
> qsub -l nodes=2,walltime=100,gres=none job.cmd
```

In this case, the job only allocates compute nodes that have no generic resources associated with them.

10.7.2.D Requesting Generic Resources Automatically within a Queue/Class

Generic resource constraints can be assigned to a queue or class and inherited by any jobs that do not have a `gres` request. This enables targeting of specific resources, automation of co-allocation requests, and other uses. To enable this, use the DEFAULT.GRES attribute of the CLASSCFG parameter as in the following example:

```
CLASSCFG[viz] DEFAULT.GRES=graphics:2
```

For each node requested by a `viz` job, also request two graphics cards.

10.7.3 Managing Generic Resource Race Conditions

A software license race condition 'window of opportunity' opens when Moab checks a license server for sufficient available licenses and closes when the user's software actually checks out the software licenses. The time between these two events can be seconds to many minutes depending on overhead factors such as node OS provisioning, job startup, licensed software startup, and so forth.

During this window, another Moab-scheduled job or a user or job external to the cluster or cloud can obtain enough software licenses that by the time the job attempts to obtain its software licenses, there are an insufficient quantity of available licenses. In such cases a job will sit and wait for the license, and while it waits it occupies but does not use resources that another job could have used. Use the `STARTDELAY` parameter to prevent such a situation.

```
GRESCFG[<license>] STARTDELAY=<window_of_opportunity>
```

With the `STARTDELAY` parameter enabled (on a per generic resource basis) Moab blocks any idle jobs requesting the same generic resource from starting until the `<window_of_opportunity>` passes. The window is defined by the customer on a per generic resource basis.

Related Topics

- GRESCFG parameter
- 10.9 Enabling Generic Events
- 10.3 Node Attributes
- 10.5 Managing Shared Cluster Resources (Floating Resources)
- 3.7.21 mnodectl - command to dynamically modify node resources

10.8 Enabling Generic Metrics

Moab allows organizations to enable generic performance metrics. These metrics allow decisions to be made and reports to be generated based on site specific environmental factors. This increases Moab's awareness of what is occurring within a given cluster environment, and allows arbitrary information to be associated with resources and the workload within the cluster. Uses of these metrics are widespread and can cover anything from tracking node temperature, to memory faults, to application effectiveness.

- Execute triggers when specified thresholds are reached
- Modify node allocation affinity for specific jobs
- Initiate automated notifications when thresholds are reached
- Display current, average, maximum, and minimum metrics values in reports and charts within Moab Cluster Manager

In this section:

10.8.1 Configuring Generic Metrics

A new generic metric is automatically created and tracked at the server level if it is reported by either a node or a job.

To associate a generic metric with a job or node, a Native Resource Manager must be set up and the GMETRIC attribute must be specified. For example, to associate a generic metric of `temp` with each node in a Torque cluster, the following could be reported by a Native Resource Manager:

```
# temperature output
node001 GMETRIC[temp]=113
node002 GMETRIC[temp]=107
node003 GMETRIC[temp]=83
node004 GMETRIC[temp]=85
...
```

Generic metrics are tracked as floating point values allowing virtually any number to be reported.

In the preceding example, the new metric, `temp`, can now be used to monitor system usage and performance or to allow the scheduler to take action should certain thresholds be reached. Some uses include the following:

- Executing triggers based on generic metric thresholds
- Adjust a node's availability for accepting additional workload
- Adjust a node's allocation priority
- Initiate admin notification of current, minimum, maximum, or average generic metric values
- Use metrics to report resource and job performance

- Use metrics to report resource and job failures
- Using job profiles to allow Moab to learn which resources best run which applications
- Tracking effective application efficiency to identify resource brown outs even when no node failure is obvious
- Viewing current and historical cluster-wide generic metric values to identify failure, performance, and usage
- Enable charging policies based on consumption of generic metrics patterns
- View changes in generic metrics on nodes, jobs, and cluster wide over time
- Submit jobs with generic metric based node-allocation requirements

Generic metric values can be viewed using checkjob, checknode, mdiag -n, mdiag -j, or Moab Cluster Manager Charting and Reporting Features.

Historical job and node generic metric statistics can be cleared using the mjobctl and mnodectl commands.

10.8.2 Example Generic Metric Usage

As an example, consider a cluster with two primary purposes for generic metrics. The first purpose is to track and adjust scheduling behavior based on node temperature to mitigate overheating nodes. The second purpose is to track and charge for utilization of a locally developed data staging service.

The first step in enabling a generic metric is to create probes to monitor and report this information. Depending on the environment, this information may be distributed or centralized. In the case of temperature monitoring, this information is often centralized by a hardware monitoring service and available via command line or an API. If monitoring a locally developed data staging service, this information might need to be collected from multiple remote nodes and aggregated to a central location. The following are popular freely available monitoring tools:

Tool	Link
Ganglia	https://ganglia.sourceforge.net
Monit	https://www.tildeslash.com/monit
Nagios	https://www.nagios.org

Once the needed probes are in place, a Native Resource Manager interface must be created to report this information to Moab. Creating a Native Resource Manager interface should be very simple, and in most cases a script similar to those found in the `$TOOLSDIR` (`$PREFIX/tools`) directory can be used as a template. For this example, we will assume centralized information and will use the resource manager script below:

```
#!/usr/bin/perl
# 'hwctl outputs information in format '<NODEID> <TEMP>'
open(TQUERY,"/usr/sbin/hwctl -q temp |");
while (<TQUERY>)
  {
  my $nodeid,$temp = split /\w+/;
  $dstage=GetDSUsage($nodeid);
  print "$nodeid GMETRIC[temp]=$temp GMETRIC[dstage]=$dstage
";
  }
```

With the script complete, the next step is to integrate this information into Moab. This is accomplished with the following configuration line:

```
RMCFG[local] TYPE=NATIVE CLUSTERQUERYURL=file://$TOOLSDIR/node.query.local.pl
...
```

Moab can now be recycled and temperature and data staging usage information will be integrated into Moab compute node reports.

If the checknode command is run, output similar to the following is reported:

```
> checknode cluster013
...
Generic Metrics:  temp=113.2,dstage=23748
...
```

Moab Cluster Manager reports full current and historical generic metric information in its visual cluster overview screen.

The next step in configuring Moab is to inform Moab to take certain actions based on the new information it is tracking. For this example, there are two purposes. The first purpose is to get jobs to avoid hot nodes when possible. This is accomplished using the `GMETRIC` attribute of the Node Allocation Priority function as in the following example:

```
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF=PRIORITY-10*GMETRIC[temp]
...
```

This simple priority function reduces the priority of the hottest nodes making such less likely to be allocated.

The example cluster is also interested in notifying admins if the temperature of a given node ever exceeds a critical threshold. This is accomplished using a trigger. The following line will send email to admins any time the temperature of a node exceeds 120 degrees:

```
NODECFG[DEFAULT] TRIGGER=atype=mail,etype=threshold,threshold=gmetric
```

```
[temp]>120,action='warning: node $OID temp high'
...
```

Related Topics

- 10.5 Managing Shared Cluster Resources (Floating Resources)

10.9 Enabling Generic Events

Generic events are used to identify failures and other occurrences that Moab or other systems must be made aware. This information may result in automated resource recovery, notifications, adjustments to statistics, or changes in policy. Generic events also have the ability to carry an arbitrary human readable message that may be attached to associated objects or passed to admins or external systems. Generic events typically signify the occurrence of a specific event as opposed to generic metrics, which indicate a change in a measured value.

Using generic events, Moab can be configured to automatically address many failures and environmental changes improving the overall performance. Some sample events that sites might be interested in monitoring, recording, and taking action on include:

- Machine Room Status:
  - Excessive Room Temperature
  - Power Failure or Power Fluctuation
  - Chiller Health
- Network File Server Status:
  - Failed Network Connectivity
  - Server Hardware Failure
  - Full Network File System
- Compute Node Status:
  - Machine Check Event (MCE)
  - Network Card (NIC) Failure
  - Excessive Motherboard/CPU Temperature
  - Hard Drive Failures

In this section:

10.9.1 Configuring Generic Events

Generic events are defined in the `moab.cfg` file and have several different configuration options. The only required option is `action`.

The full list of configurable options for generic events is shown in the following table:

Attribute	Description
ACTION	Comma-delimited list of actions to be processed when a new event is received.
ECOUNT	Number of events that must occur before launching action. Action will be launched each `<ECOUNT>` event if rearm is set.
REARM	Minimum time between events specified in `[[[DD:]HH:]MM:]SS` format.
SEVERITY	An arbitrary severity level from 1 through 4, inclusive. SEVERITY appears in the output of `mdiag -n -v -v --xml`. The severity level will not be used for any other purpose.

In this topic:

10.9.1.A Action Types

The impact of the event is controlled using the `ACTION` attribute of the `GEVENTCFG` parameter. The `ACTION` attribute is comma-delimited and can include any combination of the actions in the following table:

Value	Description
DISABLE [:<OTYPE>:<OID>]	Marks event object (or specified object) down until event report is cleared.
EXECUTE	Executes a script at the provided path. The value of `EXECUTE` is not contained in quotation marks. Arguments are allowed at the end of the path and are separated by question marks (`?`). Trigger variables (such as `$OID`) are allowed.
NOTIFY	Notifies admins of the event occurrence.
OBJECTXMLSTDIN	If the `EXECUTE` action type is also specified, this flag passes an XML description of the firing gevent to the script.
OFF	Powers off node or resource.
ON	Powers on node or resource.
PREEMPT [:<POLICY>]	Preempts workload associated with object (valid for node, job, reservation, partition, resource manager, user, group, account, class, QoS, and cluster objects).
RECORD	Records events to the event log. The record action causes a line to be added to the event log regardless of whether or not the parameter RECORDEVENTLIST includes GEVENT.
RESERVE [:<DURATION>]	Reserves node for specified duration (default: 24 hours).
RESET	Resets object (valid for nodes - causes reboot).
SIGNAL[:<SIGNO>]	Sends signal to associated jobs or services (valid for node, job, reservation, partition, resource manager, user, group, account, class, QoS, and cluster objects).

This is an example of using `objectxmlstdin` with a gevent:

```
<gevent name="bob" statuscode="0" time="1320334763">Testing</gevent>
```

10.9.1.B Named Events

In general, generic events are named, with the exception of those based on generic metrics. Names are used primarily to differentiate between different events and do not have any intrinsic meaning to Moab. It is suggested that the admin choose names that denote specific meanings within the organization.

Example 10-10: generic events

```
# Note: cpu failures require admin attention, create maintenance reservation
GEVENTCFG[cpufail] action=notify,record,disable,reserve rearm=01:00:00# Note: power
failures are transient, minimize future use
GEVENTCFG[powerfail] action=notify,record, rearm=00:05:00
# Note: fs full can be automatically fixed
GEVENTCFG[fsfull] action=notify,execute:/home/jason/MyPython/cleartmp.py?$OID?nodefix
# Note: memory errors can cause invalid job results, clear node immediately
GEVENTCFG[badmem] action=notify,record,preempt,disable,reserve
```

10.9.1.C Generic Metric (GMetric) Events

GMetric events are generic events based on generic metrics. They are used for executing an action when a generic metric passes a defined threshold. Unlike named events, GMetric events are not named and use the following format: `GEVENTCFG [GMETRIC<COMPARISON>VALUE] ACTION=...`

Example 10-11: GMetric events

```
GEVENTCFG[cputemp>150] action=off
```

This form of generic events uses the GMetric name, as returned by a `GMETRIC` attribute in a Native Resource Manager interface.

Only one generic event can be specified for any given generic metric.

Valid comparative operators are shown in the following table:

Type	Comparison	Notes
>	greater than	Numeric values only
> =	greater than or equal to	Numeric values only
= =	equal to	Numeric values only
<	less than	Numeric values only
< =	less than or equal to	Numeric values only
< >	not equal	Numeric values only

10.9.2 Reporting Generic Events

Unlike generic metrics, generic events can be optionally configured at the global level to adjust rearm policies, and other behaviors. In all cases, this is accomplished using the GEVENTCFG parameter.

To report an event associated with a job or node, use the Native Resource Manager interface or the mjobctl or mnodectl commands. You can report generic events on the scheduler with the mschedctl command.

If using the Native Resource Manager interface, use the GEVENT attribute as in the following example:

```
node001 GEVENT[hitemp]='temperature exceeds 150 degrees'
node017 GEVENT[fullfs]='/var/tmp is full'
```

The time at which the event occurred can be passed to Moab to prevent multiple processing of the same event. This is accomplished by specifying the event type in the format `<GEVENTID>[:<EVENTTIME>]` as below:

```
node001 GEVENT[hitemp:1130325993]='temperature exceeds 150 degrees'
node017 GEVENT[fullfs:1130325142]='/var/tmp is full'
```

10.9.3 Generic Events Attributes

Each node will record the following about reported generic events:

- status - is event active
- message - human readable message associated with event
- count - number of event incidences reported since statistics were cleared
- time - time of most recent event

Each event can be individually cleared, annotated, or deleted by cluster admins using a mnodectl command.

10.9.4 Manually Creating Generic Events

Generic events can be manually created on a physical node.

To add GEVENT `event` with message `"hello"` to `node02`, do the following:

```
> mnodectl -m gevent=event:"hello" node02
```

Related Topics

- 10.5 Managing Shared Cluster Resources (Floating Resources)

Chapter 11: Resource Managers and Interfaces

In this chapter:

Moab provides a powerful resource management interface that enables significant flexibility in how resources and workloads are managed. Highlights of this interface are listed below:

Highlight	Description
Support for Generic Resource Manager Interfaces	Manage cluster resources securely via locally developed or open source projects using simple flat text interfaces or XML over HTTP.
Support for Multiple Simultaneous Resource Managers	Integrate resource and workload streams from multiple independent sources reporting disjoint sets of resources.
Independent Workload and Resource Management	Allow one system to manage your workload (queue manager) and another to manage your resources.
Support for Rapid Development Interfaces	Load resource and workload information directly from a file, a URL, or from the output of a configurable script or other executable.
Resource Extension Information	Integrate information from multiple sources to obtain a cohesive view of a compute resource. That

Highlight	Description
	is, mix information from a resource manager and a cluster performance monitor to obtain a single node image with a coordinated state and a more extensive list of node configuration and utilization attributes.

11.1 Resource Manager

For most installations, Moab Workload Manager uses the services of a resource manager to obtain information about the state of compute resources (nodes) and workload (jobs). Moab also uses the resource manager to manage jobs, passing instructions regarding when, where, and how to start or otherwise manipulate jobs.

Moab can be configured to manage more than one resource manager simultaneously, even resource managers of different types. Using a local queue, jobs can even be migrated from one resource manager to another. However, there are currently limitations regarding jobs submitted directly to a resource manager (not to the local queue.) In such cases, the job is constrained to only run within the bound of the resource manager to which it was submitted.

In this section:

11.1.1 Scheduler/Resource Manager Interactions

Moab interacts with all resource managers using a common set of commands and objects. Each resource manager interfaces, obtains, and translates Moab concepts regarding workload and resources into Native Resource Manager objects, attributes, and commands.

Information on creating a new scheduler resource manager interface can be found in 11.4 Adding New Resource Manager Interfaces.

In this topic:

11.1.1.A Resource Manager Commands

For many environments, Moab interaction with the resource manager is limited to the following objects and functions:

Object	Function	Details
Job	Query	Collect detailed state, requirement, and utilization information about jobs
	Modify	Change job state and/or attributes
	Start	Execute a job on a specified set of resources
	Cancel	Cancel an existing job
	Preempt/Resume	Suspend, resume, checkpoint, restart, or requeue a job
Node	Query	Collect detailed state, configuration, and utilization information about compute resources
	Modify	Change node state and/or attributes
Queue	Query	Collect detailed policy and configuration information from the resource manager

Using these functions, Moab is able to fully manage workload, resources, and cluster policies. More detailed information about resource manager specific capabilities and limitations for each of these functions can be found in the individual resource manager overviews (PBS or WIKI).

Beyond these base functions, other commands exist to support advanced features such as provisioning and cluster level resource management.

11.1.1.B Resource Manager Flow

In general, Moab interacts with resource managers in a sequence of steps each scheduling iteration. These steps are outlined below:

1. load global resource information
2. load node specific information (optional)
3. load job information
4. load queue/policy information (optional)
5. cancel/preempt/modify jobs according to cluster policies
6. start jobs in accordance with available resources and policy constraints
7. handle user commands

Typically, each step completes before the next step is started. However, with current systems, size and complexity mandate a more advanced parallel approach providing benefits in the areas of reliability, concurrency, and responsiveness.

11.1.2 Resource Manager Specific Details (Limitations/Special Features)

- Torque
  - Torque Homepage
- Wiki
  - Wiki Overview

11.1.3 Synchronizing Conflicting Information

Moab does not trust resource manager information. Node, job, and policy information is reloaded on each iteration and discrepancies are detected. Synchronization issues and allocation conflicts are logged and handled where possible. To assist sites in minimizing stale information and conflicts, a number of policies and parameters are available:

- Node State Synchronization Policies
- Stale Data Purging (see the parameter JOBPURGETIME)
- Thread Management (preventing resource manager failures from affecting scheduler operation)
- Resource Manager Poll Interval (see the parameter RMPOLLINTERVAL)

11.1.4 Evaluating Resource Manager Availability and Performance

Each resource manager is individually tracked and evaluated by Moab. Using the mdiag -R command, you can determine how a resource manager is configured, how heavily it is loaded, what failures, if any, have occurred in the recent past, and how responsive it is to requests.

Related Topics

- 11.2 Resource Manager Configuration
- 11.3 Resource Manager Extensions

11.2 Resource Manager Configuration

In this section:

11.2.1 Defining and Configuring Resource Manager Interfaces

Moab resource manager interfaces are defined using the RMCFG parameter. This parameter allows specification of key aspects of the interface. In most cases, only the `TYPE` attribute needs to be specified and Moab determines the needed defaults required to activate and use the selected interface. In the following example, an interface to a Loadleveler resource manager is defined:

```
RMCFG[orion] TYPE=LL...
```

Note that the resource manager is given a label of `orion`. This label can be any arbitrary site-selected string and is for local usage only. For sites with multiple active resource managers, the labels can be used to distinguish between them for resource manager specific queries and commands.

Resource Manager Attributes

The following table lists the possible resource manager attributes that can be configured:

Resource Manager Attributes

ADMINEXEC	MINETIME
AUTHTYPE	MINETIME
BANDWIDTH	NMPORT
CHECKPOINTSIG	NODEFAILURERSVPROFILE
CHECKPOINTTIMEOUT	NODESTATEPOLICY
CLIENT	OMAP
CLUSTERQUERYURL	PORT
CONFIGFILE	PROVDURATION
DEFAULTCLASS	RESOURCECREATEURL
DEFAULTHIGHSPEEDADAPTER	RESOURCETYPE
DESCRIPTION	RMSTARTURL
ENV	RMSTOPURL
EPORT	SBINDIR
FAILTIME	SERVER
FBSERVER	STAGETHRESHOLD
FLAGS	STARTCMD
FNLIST	SUBMITCMD
HOST	SUBMITPOLICY
JOBCANCELURL	SUSPENDSIG
JOBEXTENDDURATION	SYNCJOBID

JOBIDFORMAT	SYSTEMMODIFYURL
JOBMODIFYURL	SYSTEMQUERYURL
JOBRSVRECREATE	TARGETUSAGE
JOBSTARTURL	TIMEOUT
JOBSUBMITURL	TRIGGER
JOBSUSPENDURL	TYPE
JOBVALIDATEURL	VARIABLES
MAXDSOP	VERSION
MAXITERATIONFAILURECOUNT	WORKLOADQUERYURL
MAXJOBPERMINUTE	

ADMINEXEC	
Format	"jobsubmit"
Default	NONE
Description	Normally, when the JOBSUBMITURL is executed, Moab will drop to the UID and GID of the user submitting the job. Specifying an ADMINEXEC of jobsubmit causes Moab to use its own UID and GID instead (usually root). This is useful for some Native Resource Managers where the JOBSUBMITURL is not a user command (such as *qsub*) but a script that interfaces directly with the resource manager.
Example	`RMCFG[base] ADMINEXEC=jobsubmit` *Moab will not use the user's UID and GID for executing the JOBSUBMITURL.*

AUTHTYPE	
Format	One of CHECKSUM, OTHER, PKI, SECUREPORT, or NONE

AUTHTYPE	
Default	`CHECKSUM`
Description	The security protocol to be used in scheduler-resource manager communication. Only valid with WIKI based interfaces.
Example	`RMCFG[base] AUTHTYPE=CHECKSUM` *Moab requires a secret key-based checksum associated with each resource manager message.*

BANDWIDTH	
Format	`<FLOAT>[{M|G|T}]`
Default	`-1` (unlimited)
Description	The maximum deliverable bandwidth between the Moab server and the resource manager for staging jobs and data. Bandwidth is specified in units per second and defaults to a unit of MB/s. If a unit modifier is specified, the value is interpreted accordingly (M - megabytes/sec, G - gigabytes/sec, T - terabytes/sec).
Example	`RMCFG[base] BANDWIDTH=340G` *Moab will reserve up to 340 GB of network bandwidth when scheduling job and data staging operations to and from this resource manager.*

CHECKPOINTSIG	
Format	One of `suspend`, `<INTEGER>` or `SIG<X>`
Description	Specifies what signal to send the resource manager when a job is checkpointed. See 9.4 Checkpoint/Restart Facilities.
Example	`RMCFG[base] CHECKPOINTSIG=SIGKILL` *Moab routes the signal `SIGKILL` through the resource manager*

CHECKPOINTSIG	
	to the job when a job is checkpointed.

CHECKPOINTTIMEOUT	
Format	`[[[DD:]HH:]MM:]SS`
Default	`0` (no timeout)
Description	Specifies how long Moab waits for a job to checkpoint before canceling it. If set to `0`, Moab does not cancel the job if it fails to checkpoint. See 9.4 Checkpoint/Restart Facilities.
Example	`RMCFG[base] CHECKPOINTTIMEOUT=5:00` *Moab cancels any job that has not exited 5 minutes after receiving a checkpoint request.*

CLIENT	
Format	`<PEER>`
Default	Use name of resource manager for peer client lookup
Description	If specified, the resource manager will use the peer value to authenticate remote connections. See 23.12.2 Configuring Peer Data Staging. If not specified, the resource manager will search for a CLIENTCFG[<X>] entry of `RM:<RMNAME>`in the `moab-private.cfg` file.
Example	`RMCFG[clusterBI] CLIENT=clusterB` Moab will look up and use information for peer `clusterB` when authenticating the `clusterBI` resource manager.

CLUSTERQUERYURL	
Format	`[<protocol>://][<path>]`
Description	Specifies how Moab queries the resource manager. See 11.5 Managing

CLUSTERQUERYURL	
	Resources Directly with the Native Resource Manager Interface and URL Notes below.
Example	`RMCFG[base] CLUSTERQUERYURL=file:///tmp/cluster.config` *Moab reads `/tmp/cluster.config` when it queries `base` resource manager.*

CONFIGFILE	
Format	`<STRING>`
Description	The resource manager specific configuration file that must be used to enable correct API communication. Only valid with LL-based interfaces. This parameter is deprecated and may be removed in a future release.
Example	`RMCFG[base] TYPE=LL CONFIGFILE=/home/loadl/loadl_config` *The scheduler uses the specified file when establishing the resource manager/scheduler interface connection.*

DEFAULTCLASS	
Format	`<STRING>`
Description	The class to use if jobs submitted via this resource manager interface do not have an associated class.
Example	`RMCFG[internal] DEFAULTCLASS=batch` *Moab assigns the class `batch` to all jobs from the resource manager `internal` that do not have a class assigned.* If you are using PBS as the resource manager, a job will never come from PBS without a class, and the default will never apply.

DEFAULTHIGHSPEEDADAPTER	
Format	`<STRING>`
Default	`sn0`
Description	The default high speed switch adapter to use when starting LoadLeveler jobs (supported in version 3.2 of LoadLeveler).
Example	`RMCFG[base]      DEFAULTHIGHSPEEDADAPTER=sn1` *The scheduler will start jobs requesting a high speed adapter on `sn1`.*

DESCRIPTION	
Format	`<STRING>`
Description	The human-readable description for the resource manager interface. If white space is used, the description should be quoted.
Example	`RMCFG[torque] DESCRIPTION='Torque RM for launching jobs'` *Moab annotates the Torque resource manager accordingly.*

ENV	
Format	Semicolon-delimited (;) list of `<KEY>=<VALUE>` pairs
Default	`MOABHOMEDIR=<MOABHOMEDIR>`
Description	Specifies a list of environment variables that will be passed to URLs of type `exec://` for that resource manager.

Example	`RMCFG[base] ENV=HOST=node001;RETRYTIME=50`
	`RMCFG[base] CLUSTERQUERYURL=exec:///opt/moab/tools/cluster.query.pl`
	`RMCFG[base] WORKLOADQUERYURL=exec:///opt/moab/tools/workload.query.pl`
	The environment variables HOST and RETRYTIME (with values `node001` and `50` respectively) are passed to the `/opt/moab/tools/cluster.query.pl` and `/opt/moab/tools/workload.query.pl` when they are executed.

EPORT	
Format	`<INTEGER>`
Description	The event port to use to receive resource manager based scheduling events.
Example	`RMCFG[base] EPORT=15017` *The scheduler will look for scheduling events from the resource manager host at port* `15017`.

FAILTIME	
Format	`[[[DD:]HH:]MM:]SS`
Description	Specifies how long a resource manager must be down before any failure triggers associated with the resource manager fire.
Example	`RMCFG[base] FAILTIME=3:00` *If the* `base` *resource manager is down for three minutes, any resource manager failure triggers fire.*

FBSERVER	
Format	`<RMNAME>`
Description	The fallback server to use when talking to Moab in an HA configuration.
Example	`RMCFG[base] TYPE=MOAB SERVER=server1  FBSERVER=server1-ha`

FLAGS	
Format	Comma-delimited list of zero or selected resource manager flags. See 11.2.2.D Resource Manager Flags for valid values.
Description	Specifies various attributes of the resource manager.
Example	`RMCFG[base] FLAGS=asyncstart`

FLAGS	
	Moab directs the resource manager to start the job asynchronously.

FNLIST	
Format	Comma-delimited list of zero or more of the following: `clusterquery`, `jobcancel`, `jobrequeue`, `jobresume`, `jobstart`, `jobsuspend`, `queuequery`, `resourcequery` or `workloadquery`
Description	By default, a resource manager utilizes all functions supported to query and control batch objects. If this parameter is specified, only the listed functions are used.
Example	`RMCFG[base] FNLIST=queuequery` *Moab only uses this resource manager interface to load queue configuration information.*

HOST	
Format	`<STRING>`
Default	`localhost`
Description	The host name of the machine on which the resource manager server is running.
Example	`RMCFG[base] host=server1`

JOBCANCELURL	
Format	[`<protocol>://`][`<path>`]
Default	---
Description	Specifies how Moab cancels jobs via the resource manager. See the section URL Notes below.

JOBCANCELURL	
Example	`RMCFG[base] JOBCANCELURL=exec:///opt/moab/job.cancel.lsf.pl` *Moab executes `/opt/moab/job.cancel.lsf.pl` to cancel specific jobs.*

JOBEXTENDDURATION	
Format	`[[[DD:]HH:]MM:]SS[,[[[DD:]HH:]MM:]SS][!][<]` (or `<MIN TIME>[,<MAX TIME>][!]`)
Default	---

Description	The minimum and maximum amount of time that can be added to a job's walltime if it is possible for the job to be extended. See the MINWCLIMIT job extension. As the job runs longer than its current specified minimum wallclock limit (-l minwclimit, for example), Moab attempts to extend the job's limit by the minimum `JOBEXTENDDURATION`. This continues until either the extension can no longer occur (it is blocked by a reservation or job), the maximum `JOBEXTENDDURATION` is reached, or the user's specified wallclock limit (-l walltime) is reached. When a job is extended, it is marked as PREEMPTIBLE, unless the `!` is appended to the end of the configuration string. If the less than sign (<) is appended to the end of the expression, the values are taken to be a per-iteration minimum and maximum extension with the overall maximum extension being the job's walltime limit.
	JOBEXTENDDURATION and JOBEXTENDSTARTWALLTIME TRUE cannot be configured together. If they are in the same moab.cfg or are both active, then the JOBEXTENDDURATION will not be honored.
	For example, comment out the JOBEXTENDSTARTWALLTIME.
	`RMCFG[base] JOBEXTENDDURATION=30,1:00:00`
	`#JOBEXTENDSTARTWALLTIME TRUE`

Example	`RMCFG[base] JOBEXTENDDURATION=30,1:00:00` *Moab extends a job's walltime by `30` seconds each time the job is about to run out of walltime until it is bound by one hour, a reservation/job, or the job's original 'maximum' wallclock limit.*

JOBIDFORMAT	
Format	INTEGER
Default	---
Description	Specifies that Moab should use numbers to create job IDs. This eliminates multiple job IDs associated with a single job.
Example	`RMCFG[base] JOBIDFORMAT=INTEGER` *Job IDs are generated as numbers.*

JOBMODIFYURL	
Format	[`<protocol>://`][`<path>`]
Default	---
Description	Specifies how Moab modifies jobs via the resource manager. See the section URL Notes below.
Example	`RMCFG[base] JOBMODIFYURL=exec://$TOOLSDIR/job.modify.dyn.pl` *Moab executes `/opt/moab/job.modify.dyn.pl` to modify specific jobs.*

JOBRSVRECREATE	
Format	Boolean
Default	TRUE
Description	Specifies whether Moab will re-create a job reservation each time job information is updated by a resource manager. See I.2.6 Reducing Job Reservation Creation Time for more information.
Example	`RMCFG[base] JOBRSVRECREATE=FALSE` *Moab only creates a job reservation once when the job first starts.*

JOBSTARTURL	
Format	`[<protocol>://][<path>]`
Default	`TRUE`
Description	Specifies how Moab starts jobs via the resource manager. See the section URL Notes below.
Example	`RMCFG[base] JOBSTARTURL=exec://$TOOLSDIR/job.submit.slurm.py` *Moab will execute the `job.submit.slurm.py` script when it wants to start a job.*

JOBSUBMITURL	
Format	`[<protocol>://][<path>]`
Description	Moab submits jobs to the script `job.submit.dyn.pl` located in the TOOLSDIR.
Example	`RMCFG[base] JOBSUBMITURL=exec://$TOOLSDIR/job.submit.dyn.pl` *Moab submits jobs directly to the database located on host `dbserver.flc.com`.*

JOBSUSPENDURL	
Format	`[<protocol>://][<path>]`
Description	Specifies how Moab suspends jobs via the resource manager. See the section URL Notes below.
Example	`RMCFG[base] JOBSUSPENDURL=EXEC://$HOME/scripts/job.suspend` *Moab executes the `job.suspend` script when jobs are suspended.*

JOBVALIDATEURL	
Format	`[<protocol>://][<path>]`

JOBVALIDATEURL	
Description	Specifies how Moab validates newly submitted jobs. See the section URL Notes below. If the script returns with a non-zero exit code, the job is rejected. See 9.7.4 Enabling Job User Proxy.
Example	`RMCFG[base] JOBVALIDATEURL=exec://$TOOLS/job.validate.pl` *Moab executes the '`job.validate.pl`' script when jobs are submitted to verify they are acceptable.*

MAXDSOP	
Format	`<INTEGER>`
Default	`-1` (unlimited)
Description	The maximum number of data staging operations that can be simultaneously active.
Example	`RMCFG[ds] MAXDSOP=16`

MAXITERATIONFAILURECOUNT	
Format	`<INTEGER>`
Default	`80`
Description	The number of times the resource manager must fail within a certain iteration before Moab considers it down or corrupt. When a resource manager is down or corrupt, Moab will not attempt to interact with it.
Example	`RMCFG[base] MAXITERATIONFAILURECOUNT=25` *The resource manager `base` must fail `25` times in a single iteration for Moab to consider it down and cease interacting with it.*

MAXJOBPERMINUTE	
Format	`<INTEGER>`
Default	`-1` (unlimited)
Description	The maximum number of jobs allowed to start per minute via the resource manager.
Example	`RMCFG[base] MAXJOBPERMINUTE=5` *The scheduler only allows five jobs per minute to launch via the resource manager* `base`*.*

MAXJOBS	
Format	`<INTEGER>`
Default	`0` (limited only by the Moab `MAXJOB` setting)
Description	The maximum number of active jobs that this interface is allowed to load from the resource manager. Only works with Moab peer resource managers at this time.
Example	`RMCFG[cluster1] SERVER=moab://cluster1 MAXJOBS=200` *The scheduler loads up to* `200` *active jobs from the remote Moab peer* `cluster1`*.*

MINETIME	
Format	`<INTEGER>`
Default	`1`
Description	The minimum time in seconds between processing subsequent scheduling events.
Example	`RMCFG[base] MINETIME=5`

MINETIME	
	The scheduler batch-processes scheduling events that occur less than five seconds apart.

NMPORT	
Format	`<INTEGER>`
Default	(any valid port number)
Description	Allows specification of the resource manager's node manager port and is only required when this port has been set to a non-default value.
Example	`RMCFG[base] NMPORT=13001` *The scheduler contacts the node manager located on each compute node at port `13001`.*

NODEFAILURERSVPROFILE	
Format	`<STRING>`
Description	The rsv template to use when placing a reservation onto failed nodes. See also the NODEFAILURERESERVETIME parameter.

Example	`# moab.cfg`
	 `RMCFG[base] NODEFAILURERSVPROFILE=long`
	 `RSVPROFILE[long]        DURATION=25:00RSVPROFILE[long]`
	 `USERLIST=john`
	 The scheduler will use the `long` rsv profile when creating reservations over failed nodes belonging to base.

NODESTATEPOLICY	
Format	One of `OPTIMISTIC` or `PESSIMISTIC`
Default	`PESSIMISTIC`
Description	Specifies how Moab should determine the state of a node when multiple

NODESTATEPOLICY	
	resource managers are reporting state: • OPTIMISTIC specifies that if any resource manager reports a state of up, that state will be used. • PESSIMISTIC specifies that if any resource manager reports a state of down, that state will be used.

Example	`# moab.cfg`
	`RMCFG[native] TYPE=NATIVE NODESTATEPOLICY=OPTIMISTIC`

OMAP	
Format	[`<protocol>://`][`<path>`]
Description	Specifies an object map file that is used to map credentials and other objects when using this resource manager peer. See 23.11 Grid Credential Management for more information.

Example	`moab.cfg`
	`RMCFG[peer1] OMAP=file:///opt/moab/omap.dat`
	When communicating with the resource manager `peer1`, objects are mapped according to the rules defined in the `/opt/moab/omap.dat` file.

PORT	
Format	`<INTEGER>`
Default	`0`
Description	The port on which the scheduler should contact the associated resource manager. The value `0` specifies that the resource manager default port should be used.
Example	`RMCFG[base] TYPE=PBS HOST=cws PORT=20001` *Moab attempts to contact the PBS server daemon on host cws, port `20001`.*

PROVDURATION	
Format	`[[[DD:]HH:]MM:]SS`
Default	`2:30`
Description	The upper bound (walltime) of a provisioning request. After this duration, Moab will consider the provisioning attempt failed.
Example	`RMCFG[base] PROVDURATION=5:00` *When resource manager `base` provisions a node for more than 5 minutes, Moab considers the provisioning as having failed.*

RESOURCECREATEURL	
Format	`[<protocol>://][<path>]`
Description	Specifies a script or method that can be used by Moab to create resources dynamically, such as creating a virtual machine on a hypervisor.
Example	`RMCFG[base] RESOURCECREATEURL=exec:///opt/script/vm.provision.py` *Moab invokes the `vm.provision.py` script, passing in data as command line arguments, to request a creation of new resources.*

RESOURCETYPE	
Format	`{COMPUTE|FS|LICENSE|NETWORK|PROV}`
Description	Specifies which type of resource this resource manager is configured to control. See 11.5 Managing Resources Directly with the Native Resource Manager Interface for more information. If LICENSE is specified, *all* generic resources reported by the resource manager will be marked as a license; causing them to be tracked by the accounting manager under the Licenses property. See Licenses for more information on the Licenses property.
Example	`RMCFG[base] TYPE=NATIVE RESOURCETYPE=FS`

RESOURCETYPE	
	Resource manager `base` will function as a `NATIVE` resource manager and control file systems.

RMSTARTURL	
Format	`[<protocol>://][<path>]`
Description	Specifies how Moab starts the resource manager.
Example	`RMCFG[base] RMSTARTURL=exec:///tmp/nat.start.pl` *Moab executes `/tmp/nat.start.pl` to start the resource manager `base`.*

RMSTOPURL	
Format	`[<protocol>://][<path>]`
Description	Specifies how Moab stops the resource manager.
Example	`RMCFG[base] RMSTOPURL=exec:///tmp/nat.stop.pl` *Moab executes `/tmp/nat.stop.pl` to stop the resource manager `base`.*

SBINDIR	
Format	`<PATH>`
Description	For use with Torque; specifies the location of the Torque system binaries.
Example	`RMCFG[base] TYPE=pbs SBINDIR=/usr/local/torque/sbin` *Moab tells Torque that its system binaries are located in `/usr/local/torque/sbin`.*

SERVER	
Format	`<URL>`
Description	The resource management service to use. If not specified, the scheduler locates the resource manager via built-in defaults or, if available, with an information service.
Example	`RMCFG[base] server=ll://supercluster.org:9705` *Moab attempts to use the Loadleveler scheduling API at the specified location.*

STAGETHRESHOLD	
Format	`[[[DD:]HH:]MM:]SS`
Description	The maximum time a job waits to start locally before considering being migrated to a remote peer. In other words, if a job's start time on a remote cluster is less than the start time on the local cluster, but the difference between the two is less than `STAGETHRESHOLD`, then the job is scheduled locally. The aim is to avoid job/data staging overhead if the difference in start times is minimal. If this attribute is used, backfill is disabled for the associated resource manager.
Example	`RMCFG[remote_cluster] STAGETHRESHOLD=00:05:00` *Moab only migrates jobs to* `remote_cluster` *if the jobs can start five minutes sooner on the remote cluster than they could on the local cluster.*

STARTCMD	
Format	`<STRING>`
Description	The full path to the resource manager job start client. If the resource manager API fails, Moab executes the specified start command in a second attempt to start the job.

STARTCMD	
	Moab calls the start command with the format `<CMD><JOBID> -H <HOSTLIST>` unless the environment variable MOABNOHOSTLIST is set, in which case Moab will only pass the job ID.
Example	`RMCFG[base] STARTCMD=/usr/local/bin/qrun` *Moab uses the specified start command if API failures occur when launching jobs.*

SUBMITCMD	
Format	`<STRING>`
Description	The full path to the resource manager job submission client.
Example	`RMCFG[base] SUBMITCMD=/usr/local/bin/qsub` *Moab uses the specified submit command when migrating jobs.*

SUBMITPOLICY	
Format	One of `NODECENTRIC` or `PROCCENTRIC`
Default	`PROCCENTRIC`
Description	If set to `NODECENTRIC`, each specified node requested by the job is interpreted as a true compute host, not as a task or processor.
Example	`RMCFG[base] SUBMITPOLICY=NODECENTRIC` *Moab uses the specified submit policy when migrating jobs.*

SUSPENDSIG	
Format	`<INTEGER>` (valid UNIX signal between 1 and 64)

SUSPENDSIG	
Default	Resource manager-specific default
Description	If set, Moab sends the specified signal to a job when a job suspend request is issued.
Example	`RMCFG[base] SUSPENDSIG=19` *Moab uses the specified suspend signal when suspending jobs within the base resource manager.* `SUSPENDSIG` should not be used with Torque or other PBS-based resource managers.

SYNCJOBID	
Format	`<BOOLEAN>`
Description	Specifies that Moab should migrate jobs to the local resource manager with the job's Moab-assigned job ID. In a grid, the grid-head will only pass dependencies to the underlying Moab if `SYNCJOBID` is set. This attribute can be used with the JOBIDFORMAT attribute and PROXYJOBSUBMISSION flag in order to assign job IDs from Moab to the resource manager. For more information about all the steps necessary for using one job ID between Moab and Torque, see the section Synchronizing Job IDs in Torque and Moab below.
Example	`RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE`

SYSTEMMODIFYURL	
Format	`[<protocol>://][<path>]`
Description	Specifies how Moab modifies attributes of the system. This interface is used in data staging.
Example	`RMCFG[base] SYSTEMMODIFYURL=exec:///tmp/system.modify.pl` *Moab executes `/tmp/system.modify.pl` when it modifies system attributes in conjunction with the resource manager `base`.*

SYSTEMQUERYURL	
Format	`[<protocol>://][<path>]`
Description	Specifies how Moab queries attributes of the system. This interface is used in data staging.
Example	`RMCFG[base] SYSTEMQUERYURL=file:///tmp/system.query` *Moab reads `/tmp/system.query` when it queries the system in conjunction with `base` resource manager.*

TARGETUSAGE	
Format	`<INTEGER>[%]`
Default	90%
Description	Amount of resource manager resources to explicitly use. In the case of a storage resource manager, indicates the target usage of data storage resources to dedicate to active data migration requests. If the specified value contains a percent sign (`%`), the target value is a percent of the configured value. Otherwise, the target value is considered to be an absolute value measured in megabytes (MB).

Example	`RMCFG[storage] TYPE=NATIVE RESOURCETYPE=storage`
	`RMCFG[storage] TARGETUSAGE=80%`
	Moab schedules data migration requests to never exceed `80%` usage of the storage resource manager's disk cache and network resources.

TIMEOUT	
Format	`<INTEGER>`
Default	`30`
Description	Time (in seconds) the scheduler waits for a response from the resource manager.
Example	`RMCFG[base] TIMEOUT=40`

TIMEOUT	
	Moab waits 40 seconds to receive a response from the resource manager before timing out and giving up. Moab tries again on the next iteration.

TRIGGER	
Format	`<TRIG_SPEC>`
Description	A trigger specification indicating behaviors to enforce in the event of certain events associated with the resource manager, including resource manager start, stop, and failure.
Example	`RMCFG[base] TRIGGER=<X>`

TYPE	
Format	`<RMTYPE>[:<RMSUBTYPE>]` Where `<RMTYPE>` is one of the following: Torque, NATIVE, PBS, RMS, SSS, or WIKI and the optional `<RMSUBTYPE>` value is one of RMS.
Default	PBS
Description	Specifies type of resource manager to be contacted by the scheduler. For TYPE WIKI, AUTHTYPE must be set to CHECKSUM. The `<RMSUBTYPE>` option is currently only used to support Compaq's RMS resource manager in conjunction with PBS. In this case, the value `PBS:RMS` should be specified.

Example	`RMCFG[clusterA] TYPE=PBS HOST=clusterA PORT=15003`
	`RMCFG[clusterB] TYPE=PBS HOST=clusterB PORT=15005`
	Moab interfaces to two different PBS resource managers, one located on server `clusterA` at port `15003` and one located on server `clusterB` at port `15005`.

VARIABLES	
Format	`<VAR>=<VAL>[,<VAR>=<VAL>]`
Description	Opaque resource manager variables.
Example	`RMCFG[base] VARIABLES=SCHEDDHOST=head1` *Moab associates the variable `SCHEDDHOST` with the value head1 on resource manager `base`.*

VERSION	
Format	`<STRING>`
Description	Resource manager-specific version string.
Example	`RMCFG[base] VERSION=10124` *Moab assumes that resource manager `base` has a version number of `1.1.24`.*

WORKLOADQUERYURL	
Format	`[<protocol>://][<path>]`
Description	Specifies how Moab queries the resource manager for workload information. See 11.5 Managing Resources Directly with the Native Resource Manager Interface and the section URL Notes below.
Example	`RMCFG[Torque] WORKLOADQUERYURL=exec://$TOOLSDIR/job.query.dyn.pl` *Moab executes `/opt/moab/tools/job.query.dyn.pl` to obtain updated workload information from resource manager `Torque`.*

URL Notes

URL parameters can load data by using the `exec` or `file` protocols.

For the protocol `file`, Moab loads the data directly from the text file pointed to by path:

```
RMCFG[base] SYSTEMQUERYURL=file:///tmp/system.query
```

For the protocol `exec`, Moab executes the file pointed to by path and loads the output written to STDOUT. If the script requires arguments, you can use a question mark (`?`) between the script name and the arguments, and an ampersand (`&`) for each space.

```
RMCFG[base] JOBVALIDATEURL=exec://$TOOLS/job.validate.pl
RMCFG[native] CLUSTERQUERYURL=exec://opt/moab/tools/cluster.query.pl?-group=group1&-
arch=x86
```

Synchronizing Job IDs in Torque and Moab

Unless you use an msub submit filter or you're in a grid, we recommend that you use your resource manager-specific job submission command (for instance, *`qsub`*).

In order to synchronize your job IDs between Torque and Moab, you must perform the following steps.

1. Set the attribute SYNCJOBID to `TRUE` in all resource managers:

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   ```

2. Set the PROXYJOBSUBMISSION flag. With `PROXYJOBSUBMISSION` enabled, you must run Moab as a Torque manager or operator. Verify that other users can submit jobs using msub. Moab, as a non-root user, should still be able to submit jobs to Torque and synchronize job IDs.

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   RMCFG[torque] FLAGS=PROXYJOBSUBMISSION
   ```

3. Add JOBIDFORMAT=`INTEGER` to the internal resource manager. Adding this parameter forces Moab to only use numbers as job IDs and those numbers to use across Moab, Torque, and the entire grid. This enhances the end-user experience as it eliminates multiple job IDs associated with a single job.

   ```
   RMCFG[torque] TYPE=PBS SYNCJOBID=TRUE
   RMCFG[torque] FLAGS=PROXYJOBSUBMISSION

   RMCFG[internal] JOBIDFORMAT=INTEGER
   ```

11.2.2 Resource Manager Configuration Details

As with all scheduler parameters, the `RMCFG` parameter follows the syntax described in 2.6 Configuring the Scheduler.

In this topic:

11.2.2.A Resource Manager Types

The `RMCFG` parameter enables the scheduler to interface to multiple types of resource managers using the `TYPE` or `SERVER` attributes. Specifying these attributes, any of the following listed resource managers can be supported.

Type	Resource Managers	Details
Moab	Moab Workload Manager	Use the Moab peer-to-peer (grid) capabilities to enable grids and other configurations. See 23.2 Grid Configuration Basics.
MWS	Moab Web Services	The MWS resource manager type is a native integration between Moab and MWS. Resource manager data is passed directly between Moab and MWS using JSON (rather than Moab's native WIKI syntax). This simplifies resource manager configuration for systems where one or more MWS plug-ins are acting as resource managers. See the 'Moab Workload Manager Resource Manager Integration' section in the *Moab Web Services Administrator Guide* for more information.
Native	Moab *Native* Interface	Used for connecting directly to scripts, files, and databases. See 11.5 Managing Resources Directly with the Native Resource Manager Interface.
PBS	Torque (all versions)	N/A
SSS	Scalable Systems Software Project version 2.0 and higher	N/A
WIKI	Wiki interface specification	Used for most non-PBS resource managers.

Type	Resource Managers	Details
	version 1.0 and higher	

11.2.2.B Resource Manager Name

Moab can support more than one resource manager simultaneously. Consequently, the `RMCFG` parameter takes an index value such as `RMCFG[clusterA]`. This index value essentially names the resource manager (as done by the deprecated parameter `RMNAME`). The resource manager name is used by the scheduler in diagnostic displays, logging, and in reporting resource consumption to the accounting manager. For most environments, the selection of the resource manager name can be arbitrary.

11.2.2.C Resource Manager Location

The `HOST`, `PORT`, and `SERVER` attributes can be used to specify how the resource manager should be contacted. For many resource managers the interface correctly establishes contact using default values. These parameters need only to be specified for resource managers such as the WIKI interface (that do not include defaults) or with resources managers that can be configured to run at non-standard locations (such as PBS). In all other cases, the resource manager is automatically located.

11.2.2.D Resource Manager Flags

The `FLAGS` attribute can be used to modify many aspects of a resource manager's behavior.

AUTOSYNC, COLLAPSEDVIEW, HOSTINGCENTER, PRIVATE, REPORT, SHARED, and STATIC are deprecated.

Flag	Description
ASYNCDELETE	Moab directs the resource manager to not wait for confirmation that the job correctly cancels before the API call returns. See Appendix I: Considerations for Large Clusters for more information. This flag is only applicable for Torque or Moab Native Resource Managers.

Flag	Description
ASYNCSTART	Jobs started on this resource manager start asynchronously. In this case, the scheduler does not wait for confirmation that the job correctly starts before proceeding. See Appendix I: Considerations for Large Clusters for more information. This flag is only applicable for Torque or Moab Native Resource Managers.
AUTOSTART	Jobs staged to this resource manager do not need to be explicitly started by the scheduler. The resource manager itself handles job launch.
BECOMEMASTER	Nodes reported by this resource manager will transfer ownership to this resource manager if they are currently owned by another resource manager that does not have this flag set.
CLIENT	A client resource manager object is created for diagnostic/statistical purposes or to configure Moab's interaction with this resource manager. It represents an external entity that consumes server resources or services, allows a local admin to track this usage, and configures specific policies related to that resource manager. A client resource manager object loads no data and provides no services.
CLOCKSKEWCHECKING	Enables you to configure clock skew adjustments. Most of the time it is sufficient to use an NTP server to keep the clocks in your system synchronized.
DYNAMICCRED	The resource manager creates credentials within the cluster as needed to support workload.
EnableCondensedQuery	Enables the condensed workload query. Only applies if the Torque parameter `job_full_report_time` is used. See 'Server Parameters' in the *Torque Resource Manager Administrator Guide*.
EXECUTIONSERVER	The resource manager is capable of launching and executing batch workload.
FSISREMOTE	Add this flag if the working file system doesn't exist on the server to prevent Moab from validating files and directories

Flag	Description
	at migration.
FULLCP	Always checkpoint full job information (useful with Native Resource Managers).
IgnOS	Ignore the operating system reported by the resource manager on each node and use the OS in Moab's configuration files. See the node attribute OS for more information.
IGNQUEUESTATE	The queue state reported by the resource manager should be ignored. Can be used if queues must be disabled inside of a particular resource manager to allow an external scheduler to properly operate.
IGNWORKLOADSTATE	When this flag is applied to a Native Resource Manager, any jobs that are reported via that resource manager's 'workload query URL' have their reported state ignored. For example, if a resource manager has the `IgnWorkloadState` flag and it reports that a set of jobs have a state of 'Running,' this state is ignored and the jobs will either have a default state set or will inherit the state from another resource manager reporting on that same set of jobs. This flag only changes the behavior of resource managers of type `NATIVE`.
LOCALWORKLOADEXPORT	When set, destination peers share information about local and remote jobs, allowing job management of different clusters at a single peer. For more information, see 23.7 Workload Submission and Control.
MIGRATEALLJOBATTRIBUTES	When set, this flag causes additional job information to be migrated to the resource manager; additional job information includes things such as node features applied via `CLASSCFG[name] DEFAULT.FEATURES`, the account to which the job was submitted, job walltime limit, and node exclusivity.
NOAUTORES	If the resource manager does not report CPU usage to Moab because CPU usage is at 0%, Moab assumes full CPU usage. When set, Moab recognizes the resource manager report as 0% usage. This is only valid for PBS.

Flag	Description
NoCondensedQuery	Disables the condensed workload query. This is the default. Only applies if the Torque parameter `job_full_report_time` is used. See 'Server Parameters' in the *Torque Resource Manager Administrator Guide*.
NOCREATERESOURCE	To use resources discovered from this resource manager, they must be created by another resource manager first. For example, if you set `NOCREATERESOURCE` on resource manager `A`, which reports nodes 1 and 2, and resource manager `B` only reports node 1, then node 2 will not be created because resource manager `B` did not report it.
PROXYJOBSUBMISSION	Enables Admin proxy job submission, which means admins can submit jobs in behalf of other users.
PUSHSLAVEJOBUPDATES	Enables job changes made on a Moab Grid Member to be pushed to the Moab Grid Control. Without this flag, jobs being reported to the Moab Grid Control do not show any changes made on the remote Moab server (via mjobctl and so forth).
RECORDGPUMETRICS	Enables the recording of GPU metrics for nodes.
RECORDMICMETRICS	Enables the recording of MIC metrics for nodes.
SLAVEPEER	Information from this resource manager cannot be used to identify new jobs or nodes. Instead, this information can only be used to update jobs and nodes discovered and loaded from other non-slave resource managers.
THREADEDQUERIES	When this flag is set for an individual resource manager, the queries that Moab performs to get information from the resource manager is done in a separate thread from the main Moab process. This enables Moab to remain responsive during the query and ultimately reduces the time spent in a scheduling cycle. If multiple resource managers are being used, the effect can be more significant because all resource managers will be queried in parallel.
USEPHYSICALMEMORY	Tells Moab to use a node's physical memory instead of the swap space. For example, if a node has 12 GB of RAM and an additional

Flag	Description
	12 GB of swap space, it has 24 GB of virtual memory. If a 4 GB job is assigned to that node, the reported available memory shows 12 GB because the job is using the swap space not the physical memory. The reported available memory doesn't decrease until the swap space is used up. When this flag is set, the 4 GB job immediately reduces the available memory to 8 GB (physical memory - used memory).
USERSPACEISSEPARATE	Tells Moab to ignore validating the user's UID and GID in the case that information doesn't exist on the Moab server.

Example

```
# resource manager 'torque' should use asynchronous job start
RMCFG[torque] FLAGS=asyncstart
```

11.2.3 Scheduler/Resource Manager Interactions

In the simplest configuration, Moab interacts with the resource manager using the following four primary functions:

Function	Description
GETJOBINFO	Collect detailed state and requirement information about idle, running, and recently completed jobs.
GETNODEINFO	Collect detailed state information about idle, busy, and defined nodes.
STARTJOB	Immediately start a specific job on a particular set of nodes.
CANCELJOB	Immediately cancel a specific job regardless of job state.

Using these four simple commands, Moab enables nearly its entire suite of scheduling functions. More detailed information about resource manager specific requirements and semantics for each of these commands can be found in the specific resource manager (such as WIKI) overviews.

In addition to these base commands, other commands are required to support advanced features such as suspend/resume, gang scheduling, and scheduler initiated checkpoint restart.

Information on creating a new scheduler resource manager interface can be found in the section Adding New Resource Manager Interfaces.

11.3 Resource Manager Extensions

In this section:

All resource managers are not created equal. There is a wide range in what capabilities are available from system to system. Additionally, there is a large body of functionality that many, if not all, resource managers have no concept of. A good example of this is job QoS. Since most resource managers do not have a concept of quality of service, they do not provide a mechanism for users to specify this information. In many cases, Moab is able to add capabilities at a global level. However, a number of features require a *per job* specification. Resource manager extensions allow this information to be associated with the job.

11.3.1 Resource Manager Extension Specification

Specifying resource manager extensions varies by resource manager. Torque and Wiki each allow the specification of an *extension* field as described in the following table:

Resource Manager	Specification Method
Torque 2.0+	-l `> qsub -l nodes=3,qos=high sleepy.cmd`
Torque 1.x	-W x= `> qsub -l nodes=3 -W x=qos:high sleepy.cmd`
Wiki	comment `comment=qos:high`

11.3.2 Resource Manager Extension Values

All of the following job extensions will work with `msub -l` (or `msub -W x=. . .`). However, `qsub -l` only provides legacy support for a subset of these extensions; see 'Requesting Resources' in the *Torque Resource Manager Administrator Guide* for the list.

If your configuration primarily uses `qsub` to submit jobs, we recommend that you use the `qsub -W x=` syntax for all submissions with Moab job extensions to avoid `qsub` rejection for any unsupported (non-legacy) extensions.

The following job extensions are supported when using the resource manager-specific method:

Resource Manager Extension Values

ADVRES	MEM	PROLOGUE
CPUCLOCK	MICs	PVMEM
DDISK	MINPREEMPTTIME	QoS
DEADLINE	MINPROCSPEED	QUEUEJOB
DEPEND	MINWCLIMIT	REQATTR
DMEM	MSTAGEIN	RESFAILPOLICY
EPILOGUE	MSTAGEOUT	RMTYPE
EXCLUDENODES	NACCESSPOLICY	SIGNAL
FEATURE	NALLOCPOLICY	GRES and SOFTWARE
GATTR	NCPUS	SPRIORITY
GEOMETRY (Loadleveler)GMETRIC	NMATCHPOLICY	TEMPLATE
GPUs	NODESET	TERMTIME
GRES and SOFTWARE	NODESETCOUNT	TPN
HOSTLIST	NODESETDELAY	TRIG
JGROUP	NODESETISOPTIONAL	TRL (Format 1)
JOBFLAGS (a.k.a. FLAGS)	OPSYS	TRL (Format 2)
JOBREJECTPOLICY	PARTITION	VAR
MAXMEM	PMEM	VC
MAXPROC	PREF	VMEM
	PROCS	

ADVRES	
Format	`[!]<RSVID>`
Description	Specifies that reserved resources are required to run the job. If `<RSVID>` is specified, then only resources within the specified reservation can be allocated (see 6.1.1.D Job to Reservation Binding). You can request to not use a specific reservation by using `advres=!<reservationname>`.

Example	`> qsub -l advres=grid.3`
	Resources for the job must come from `grid.3`*.*
	`> qsub -l advres=!grid.5`
	Resources for the job must not come from `grid.5`

CPUCLOCK	
Format	`<STRING>`

CPUCLOCK	
Description	Specify the CPU clock frequency for each node requested for this job. A `cpuclock` request applies to every processor on every node in the request. Specifying varying CPU frequencies for different nodes or different processors on nodes in a single job request is not supported. Not all CPUs support all possible frequencies or ACPI states. If the requested frequency is not supported by the CPU, the nearest frequency is used. If a job does not place any load on the node then some OSs will drop the frequency below the requested frequency. Using `cpuclock` sets `NODEACCESSPOLICY` to `SINGLEJOB`. The clock frequency can be specified via: • a number that indicates the clock frequency (with or without the SI unit suffix). • a Linux power governor policy name. The governor names are: ◦ `performance`: This governor instructs Linux to operate each logical processor at its maximum clock frequency. This setting consumes the most power and workload executes at the fastest possible speed. ◦ `powersave`: This governor instructs Linux to operate each logical processor at its minimum clock frequency. This setting executes workload at the slowest possible speed. This setting does not necessarily consume the least amount of power since applications execute slower, and may actually consume more energy because of the additional time needed to complete the workload's execution. ◦ `ondemand`: This governor dynamically switches the logical processor's clock frequency to the maximum value when system load is high and to the minimum value when the system load is low. This setting causes workload to execute at the fastest possible speed or the slowest possible speed, depending on OS load. The system switches between consuming the most power and the least power.

CPUCLOCK	
	The power saving benefits of `ondemand` might be non-existent due to frequency switching latency if the system load causes clock frequency changes too often. This has been true for older processors since changing the clock frequency required putting the processor into the C3 'sleep' state, changing its clock frequency, and then waking it up, all of which required a significant amount of time. Newer processors, such as the Intel Xeon E5-2600 Sandy Bridge processors, can change clock frequency dynamically and much faster. ◦ `conservative`: This governor operates similar to the `ondemand` governor but is more conservative in switching between frequencies. It switches more gradually and uses all possible clock frequencies. This governor can switch to an intermediate clock frequency if it seems appropriate to the system load and usage, which the `ondemand` governor does not do. • an ACPI performance state (or P-state) with or without the P prefix. P-states are a special range of values (0-15) that map to specific frequencies. Not all processors support all 16 states, however, they all start at `P0`. `P0` sets the CPU clock frequency to the highest performance state, which runs at the maximum frequency. `P15` sets the CPU clock frequency to the lowest performance state, which runs at the lowest frequency. When reviewing job or node properties when `cpuclock` was used, be mindful of unit conversion. The OS reports frequency in Hz, not MHz or GHz. If a job does not place any load on the node then some OSs will drop the frequency below the requested frequency.

Example	`msub -l cpuclock=1800,nodes=2 script.sh`
	`msub -l cpuclock=1800mhz,nodes=2 script.sh`
	
	This job requests 2 nodes and specifies their CPU frequencies should be set to 1800 MHz.
	
	`msub -l cpuclock=performance,nodes=2 script.sh`
	
	This job requests 2 nodes and specifies their CPU frequencies should be set to the performance power governor policy.
	
	`msub -l cpuclock=3,nodes=2 script.sh`

CPUCLOCK	
	`msub -l cpuclock=p3,nodes=2 script.sh` *This job requests 2 nodes and specifies their CPU frequencies should be set to a performance state of 3.*

DDISK	
Format	`<INTEGER>`
Default	`0`
Description	Dedicated disk per task in MB.
Example	`> qsub -l ddisk=2000`

DEADLINE	
Format	Relative time: `[[[DD:]HH:]MM:]SS` Absolute time: `hh:mm:ss_mm/dd/yy`
Description	Either the relative completion deadline of job (from job submission time) or an absolute deadline where you specify the date and time the job will finish.
Example	`> qsub -l deadline=2:00:00,nodes=4 /tmp/bio3.cmd` *The job's deadline is 2 hours after its submission.*

DEPEND	
Format	`[<DEPENDTYPE>:][{jobname|jobid}.]<ID>[:[{jobname|jobid}.]<ID>]...`
Description	Allows specification of job dependencies for compute or system jobs. If no ID prefix (jobname or jobid) is specified, the ID value is interpreted as a job ID. See 9.5 Job Dependencies for more information.

DEPEND	

Example	`# submit job which will run after job 1301 and 1304 complete`
	`> msub -l depend=orion.1301:orion.1304 test.cmd`
	`orion.1322`
	`# submit jobname-based dependency job`
	`> msub -l depend=jobname.data1005 dataetl.cmd`
	`orion.1428`

DMEM	
Format	`<INTEGER>`
Default	`0`
Description	Dedicated memory per task in bytes.
Example	`> msub -l dmem=20480` *Moab will dedicate 20 MB of memory to the task.*

EPILOGUE	
Format	`<STRING>`
Description	Specifies a user owned epilogue script, which is run before the system epilogue and `epilogue.user` scripts at the completion of a job. The syntax is `epilogue=<file>`. The file can be designated with an absolute or relative path. This parameter works only with Torque.
Example	`> msub -l epilogue=epilogue_script.sh job.sh`

EXCLUDENODES	
Format	`{<nodeid>|<node_range>}[:...]`
Description	Specifies nodes that should not be considered for the given job.

EXCLUDENODES	
	Moab does not support the combination of `msub -l excludenodes` and *ENABLEHIGHTHROUGHPUT TRUE*.
Example	`> msub -l excludenodes=k1:k2:k[5-8]`

FEATURE	
Format	`<FEATURE>[{:|}<FEATURE>]...`
Description	Required list of node attribute/node features. If the *pipe* (|) character is used as a delimiter, the features are logically ORed together and the associated job can use resources that match any of the specified features. Requesting node names as features will result in the job being blocked from running.

Example	`> qsub -l feature='fastos:bigio' testjob.cmd`
	Submits testjob.cmd with fastos:bigio as a required feature.
	`> qsub -l feature=\!bigmem testjob.cmd`
	`> qsub -l feature='!bigmem' testjob.cmd`
	Submits testjob.cmd with a requirement that bigmem is ***not*** *a node feature. (The exclamation point must either be escaped (`\!bigmem`) or quoted (`'!bigmem'`).*

GATTR	
Format	`<STRING>`
Description	Generic job attribute associated with job. The maximum size for an attribute is 63 bytes (the core Moab size limit of 64, including a null byte).
Example	`> qsub -l gattr=bigjob`

GEOMETRY (Loadleveler)	
Format	`{(<TASKID>[,<TASKID>[,...]])[(<TASKID>[,...])...]}`
Description	Explicitly specified task geometry for Loadleveler.
Example	`> qsub -l nodes=2:ppn=4 -W x=geometry:'{(0,1,4,5)(2,3,6,7)}' quanta2.cmd` *The job* `quanta2.cmd` *runs tasks* `0`*,* `1`*,* `4`*, and* `5` *on one node, while tasks* `2`*,* `3`*,* `6`*, and* `7` *run on another node.*

GEOMETRY (topology-aware scheduling)	
Format	One of `<shape>, <shape>@<location>, or @<location>` • A shape is expressed as XxYxZ. • A location is an origin in Gemini coordinates expressed as X.Y.Z. • Multiple geometries can be given (separated by /).
Description	Explicitly specified task geometry for topology-aware scheduling.

Example	`> msub -l geometry=3x3x3 ...`
	`> msub -l geometry=4x1x4@12.0.0 ...`
	`> msub -l geometry=@0.4.0 ...`
	`> msub -l geometry=8x4x8/6x4x10 ...`

GMETRIC	
Format	Generic metric requirement for allocated nodes where the requirement is specified using the format `<GMNAME>[:{lt:,le:,eq:,ge:,gt:,ne:}<VALUE>]`
Description	Indicates generic constraints that must be found on all allocated nodes. If a `<VALUE>` is not specified, the node must simply possess the generic metric (see 10.8 Enabling Generic Metrics for more information).
Example	`> qsub -l gmetric=bioversion:ge:133244 testj.txt`

GPUs	
Format	`msub -l nodes=<VALUE>:ppn=<VALUE>:gpus=<VALUE>[:mode][:reseterr]` Where `mode` is one of: • *exclusive* - The default setting. The GPU is used exclusively by one process thread. • *exclusive_thread* - The GPU is used exclusively by one process thread. • *exclusive_process* - The GPU is used exclusively by one process regardless of process thread. If present, `reseterr` resets the ECC memory bit error counters. This only resets the volatile error counts, or errors since the last reboot. The permanent error counts are not affected. Moab passes the `mode` and `reseterr` portion of the request to Torque for processing. Moab does not support requesting GPUs as a GRES. Submitting `msub -l gres=gpus:x` does not work.
Description	Moab schedules GPUs as a special type of node-locked consumable generic resources. When Torque reports GPUs to Moab, Moab can schedule jobs and correctly assign GPUs to ensure that jobs are scheduled efficiently. To have Moab schedule GPUs, configure them in Torque then submit jobs using the 'GPU' attribute. Moab automatically parses the 'GPU' attribute and assigns them in the correct manner. For information about GPU metrics, see 20.4 GPU Metrics.

Example	`> msub -l nodes=2:ppn=2:gpus=1:exclusive_process:reseterr`
	Submits a job that requests 2 tasks, 2 processors and 1 GPU per task (2 GPUs total). Each GPU runs only threads related to the task and resets the volatile ECC memory big error counts at job start time.
	`> msub -l nodes=4:gpus=1,tpn=2`
	Submits a job that requests 4 tasks, 1 GPU per node (4 GPUs total), and 2 tasks per node. Each GPU is dedicated exclusively to one task process and the ECC memory bit error counters are not reset.
	`> msub -l nodes=4:gpus=1:reseterr`
	Submits a job that requests 4 tasks, 1 processor and 1 GPU per

GPUs	
	task (4 GPUs total). Each GPU is dedicated exclusively to one task process and resets the volatile ECC memory bit error counts at job start time. `> msub -l nodes=4:gpus=2+1:ppn=2,walltime=600` *Submits a job that requests two different types of tasks, the first is 4 tasks, each with 1 processor and 2 gpus, and the second is 1 task with 2 processors. Each GPU is dedicated exclusively to one task process and the ECC memory bit error counters are not reset.*

GRES and SOFTWARE	
Format	Percent sign (`%`) delimited list of generic resources where each resource is specified using the format `<RESTYPE>[{+|:}<COUNT>]`
Description	Indicates generic resources required by the job. If the generic resource is node-locked, it is a per-task count. If a `<COUNT>` is not specified, the resource count defaults to 1.
Example	`> qsub -W x=GRES:tape+2%matlab+3 testj.txt` When specifying more than one generic resource with -l, use the percent (%) character to delimit them. `> qsub -l gres=tape+2%matlab+3 testj.txt` `> qsub -l software=matlab:2 testj.txt`

HOSTLIST	
Format	Comma (`,`) or plus (`+`) delimited list of hostnames. Ranges and regular expressions are supported in `msub` only.
Description	Indicates an *exact set*, *superset*, or *subset* of nodes on which the job must run. Use the caret (`^`) or asterisk (`*`) characters to specify a host list as *superset* or *subset* respectively. An exact set is defined without a caret or asterisk. An exact set means *all* the hosts in the specified hostlist must be selected for the job. A subset means the specified hostlist is used first to select hosts for the job.

HOSTLIST	
	If the job requires more hosts than are in the subset hostlist, they will be obtained from elsewhere if possible. If the job does not require all of the nodes in the subset hostlist, it will use only the ones it needs. A superset means the hostlist is the *only* source of hosts that should be considered for running the job. If the job can't find the necessary resources in the superset hostlist it should *not* run. No other hosts should be considered in allocating the job.

Example	`> msub -l hostlist=nodeA+nodeB+nodeE`
	`hostlist=foo[1-5]`
	This is an exact set of (foo1,foo2,...,foo5). The job must run on all these nodes.
	`hostlist=foo1+foo[3-9]`
	This is an exact set of (foo1,foo3,foo4,...,foo9). The job must run on all these nodes.
	`hostlist=foo[1,3-9]`
	This is an exact set of the same nodes as the previous example.
	`hostlist=foo[1-3]+bar[72-79]`
	This is an exact set of (foo1,foo2,foo3,bar72,bar73,...,bar79). The job must run on all these nodes.
	`hostlist=^node[1-50]`
	This is a superset of (node1,node2,...,node50). These are the only nodes that can be considered for the job. If the necessary resources for the job are not in this hostlist, the job is not run. If the job does not require all the nodes in this hostlist, it will use only the ones that it needs.
	`hostlist=*node[15-25]`
	This is a subset of (node15,node16,...,node25). The nodes in this hostlist are considered first for the job. If the necessary resources for the job are not in this hostlist, Moab tries to obtain the necessary resources from elsewhere. If the job does not require all the nodes in this hostlist, it will use only the ones that it needs.

JGROUP	
Format	`<JOBGROUPID>`
Description	ID of job group to which this job belongs (different from the GID of the user running the job).
Example	`> msub -l JGROUP=bluegroup`

JOBFLAGS (a.k.a. FLAGS)	
Format	One or more of the following colon-delimited job flags including ADVRES [:RSVID], NOQUEUE, NORMSTART, PREEMPTEE, PREEMPTOR, RESTARTABLE, or SUSPENDABLE (see 2.8 Job Flags for a complete listing).
Description	Associates various flags with the job.
Example	`> qsub -l nodes=1,walltime=3600,jobflags=advres myjob.py`

JOBREJECTPOLICY	
Format	One or more of `CANCEL`, `HOLD`, `IGNORE`, `MAIL`, or `RETRY`
Default	`HOLD`
Details	The action to take when the scheduler determines that a job can never run: `CANCEL` issues a call to the resource manager to cancel the job. `HOLD` places a batch hold on the job preventing the job from being further evaluated until released by an admin. Admins can dynamically alter job attributes and possibly *fix* the job with mjobctl -m. `IGNORE` allows the job to exist within the resource manager queue but will neither process it nor report it. `MAIL` sends email to both the admin and the user when rejected jobs are detected. `RETRY` allows the job to remain idle and will only attempt to start the job when the policy violation is resolved. Any combination of attributes can be specified. This is a per-job policy specified with msub -l. JOBREJECTPOLICY also exists

JOBREJECTPOLICY	
	as a global parameter. Also see the parameter QOSREJECTPOLICY.
Example	`> msub -l jobrejectpolicy=cancel:mail`

MAXMEM	
Format	`<INTEGER>` (in megabytes)
Description	Maximum amount of memory the job can consume across all tasks before the JOBMEM action is taken.
Example	`> qsub -l x=MAXMEM:1000mb bw.cmd` *If a RESOURCELIMITPOLICY is set for per-job memory utilization, its action will be taken when this value is reached.*

MAXPROC	
Format	`<INTEGER>`
Description	Maximum CPU load the job can consume across all tasks before the JOBPROC action is taken.
Example	`> qsub -W x=MAXPROC:4 bw.cmd` *If a RESOURCELIMITPOLICY is set for per-job processor utilization, its action will be taken when this value is reached.*

MEM	
Format	`<INTEGER>`
Description	Specify the maximum amount of physical memory used by the job. If you do not specify MB or GB, Moab uses bytes if your resource manager is Torque, and MB if your resource manager is Native.
Example	`> msub -l nodes=4:ppn=2,mem=1024mb`

MEM	
	The job must have 4 compute nodes with 2 processors per node. The job is limited to 1024 MB of memory.

MICs	
Format	`msub -l nodes=<VALUE>:ppn=<VALUE>:mics=<VALUE>[:mode]` Where `mode` is one of: • *exclusive* - The default setting. The MIC is used exclusively by one process thread. • *exclusive_thread* - The MIC is used exclusively by one process thread. • *exclusive_process* - The MIC is used exclusively by one process regardless of process thread. Moab passes the `mode` portion of the request to Torque for processing. Moab does not support requesting MICs as a GRES. Submitting `msub -l gres=mics:x` does not work.
Description	Moab schedules MICs as a special type of node-locked generic resources. When Torque reports MICs to Moab, Moab can schedule jobs and correctly assign MICs to ensure that jobs are scheduled efficiently. To have Moab schedule MICs, configure them in Torque then submit jobs using the 'MIC' attribute. Moab automatically parses the 'MIC' attribute and assigns them in the correct manner.

Example	`> msub -l nodes=2:ppn=2:mics=1:exclusive_process`
	Submits a job that requests 2 tasks, 2 processors and 1 MIC per task (2 MICs total). Each MIC runs only threads related to the task.
	`> msub -l nodes=4:mics=1,tpn=2`
	Submits a job that requests 4 tasks, 1 MIC per node (4 MICs total), and 2 tasks per node. Each MIC is dedicated exclusively to one task process.
	`> msub -l nodes=4:mics=1`
	Submits a job that requests 4 tasks, 1 processor and 1 MIC per task (4 MICs total). Each MIC is dedicated exclusively to one task

MICs	
	process. `> msub -l nodes=4:mics=2+1:ppn=2,walltime=600` *Submits a job that requests two different types of tasks, the first is 4 tasks, each with 1 processor and 2 MICs , and the second is 1 task with 2 processors. Each MIC is dedicated exclusively to one task process.*

MINPREEMPTTIME	
Format	`[[DD:]HH:]MM:]SS`
Description	Minimum time job must run before being eligible for preemption. Can only be specified if associated QoS allows per-job preemption configuration by setting the preemptconfig flag.
Example	`> qsub -l minpreempttime=900 bw.cmd` *Job cannot be preempted until it has run for 15 minutes.*

MINPROCSPEED	
Format	`<INTEGER>`
Default	0
Description	Minimum processor speed (in MHz) for every node that this job will run on.
Example	`> qsub -W x=MINPROCSPEED:2000 bw.cmd` *Every node that runs this job must have a processor speed of at least 2000 MHz.*

MINWCLIMIT	
Format	`[[DD:]HH:]MM:]SS`

MINWCLIMIT	
Default	---
Description	Minimum wallclock limit job must run before being eligible for extension (see the resource manager attribute JOBEXTENDDURATION or the parameter JOBEXTENDSTARTWALLTIME).
Example	`> qsub -l minwclimit=300,walltime=16000 bw.cmd` *Job will run for at least `300` seconds but up to `16,000` seconds if possible (without interfering with other jobs).*

MSTAGEIN	
Format	`[<SRCURL>[|<SRCRUL>...]%]<DSTURL>`
Description	Indicates a job has data staging requirements. The source URL(s) listed will be transferred to the execution system for use by the job. If more than one source URL is specified, the destination URL must be a directory. The format of `<SRCURL>` is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is local. The format of `<DSTURL>` is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is remote. PROTO can be any of the following protocols: ssh, file, or gsiftp. HOST is the name of the host where the file resides. PATH is the path of the source or destination file. The destination path can be a directory when sending a single file and must be a directory when sending multiple files. If a directory is specified, it must end with a forward slash (`/`). Valid variables include: $JOBID $HOME - Path the script was run from $RHOME - Home dir of the user on the remote system $SUBMITHOST $DEST - This is the Moab where the job will run $LOCALDATASTAGEHEAD If no destination is given, the protocol and file name will be set to the same as the source.

MSTAGEIN	
	The $RHOME (remote home directory) variable is for when a user's home directory on the compute node is different than on the submission host.
Example	`> msub -Wx='mstagein=file://$HOME/helperscript.sh|file:///home/dev/datafile.txt%ssh://host/home/dev/' script.sh` Copy helperscript.sh and datafile.txt from the local machine to /home/dev/ on host for use in execution of script.sh. $HOME is a path containing a preceding / (i.e., /home/adaptive).

MSTAGEOUT	
Format	`[<SRCURL>[|<SRCRUL>...]%]<DSTURL>`
Description	Indicates whether a job has data staging requirements. The source URLs listed will be transferred from the execution system after the completion of the job. If more than one source URL is specified, the destination URL must be a directory. The format of `<SRCURL>` is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is remote. The format of `<DSTURL>` is: `[PROTO://][HOST][:PORT]][/PATH]` where the path is local. PROTO can be any of the following protocols: ssh, file, or gsiftp. HOST is the name of the host where the file resides. PATH is the path of the source or destination file. The destination path can be a directory when sending a single file and must be a directory when sending multiple files. If a directory is specified, it must end with a forward slash (/). Valid variables include: $JOBID $HOME - Path the script was run from $RHOME - Home dir of the user on the remote system $SUBMITHOST $DEST - This is the Moab where the job will run $LOCALDATASTAGEHEAD If no destination is given, the protocol and file name will be set to the same as the source.

MSTAGEOUT	
	The $RHOME (remote home directory) variable is for when a user's home directory on the compute node is different than on the submission host.
Example	`> msub -W x='mstageout=ssh://$DEST/$HOME/resultfile1.txt|ssh://host/home/dev/resultscript.sh%file:///home/dev/' script.sh` *Copy `resultfile1.txt` and `resultscript.sh` from the execution system to `/home/dev/` after the execution of script.sh is complete. $HOME is a path containing a preceding `/` (i.e., `/home/adaptive`).*

NACCESSPOLICY	
Format	One of SHARED, SINGLEJOB, SINGLETASK, SINGLEUSER, or UNIQUEUSER
Description	Specifies how node resources should be accessed. See 4.3 Node Access Policies for more information. The `naccesspolicy` option can only be used to make node access more constraining than is specified by the system, partition, or node policies. If the effective node access policy is `shared`, `naccesspolicy` can be set to `singleuser`, if the effective node access policy is `singlejob`, `naccesspolicy` can be set to `singletask`.

Example	`> qsub -l naccesspolicy=singleuser bw.cmd`
	`> bsub -ext naccesspolicy=singleuser lancer.cmd`
	Job can only allocate free nodes or nodes running jobs by same user.
	`> qsub -l naccesspolicy=singlejob jobscript.sh`
	`# OR`
	`> qsub -W x=naccesspolicy:singlejob jobscript.sh`
	Jobs can only run on specific nodes; regardless if the machine has free cores.

NALLOCPOLICY	
Format	One of the valid settings for the parameter NODEALLOCATIONPOLICY
Description	Specifies how node resources should be selected and allocated to the job. See 4.2 Node Allocation Policies for more information.
Example	`> qsub -l nallocpolicy=minresource bw.cmd` *Job should use the `minresource` node allocation policy.*

NCPUS	
Format	`<INTEGER>`
Description	The number of processors in one task where a task cannot span nodes. If `NCPUS` is used, then the resource manager's SUBMITPOLICY should be set to `NODECENTRIC` to get correct behavior. `-l ncpus=<#>` is equivalent to `-l nodes=1:ppn=<#>`when `JOBNODEMATCHPOLICY` is set to `EXACTNODE`. `NCPUS` is used when submitting jobs to an SMP. When using GPUs to submit to an SMP, use `-l ncpus=<#>:GPUs=<#>`. You cannot request both ncpus and nodes in the same job.

NMATCHPOLICY	
Format	One of the valid settings for the parameter JOBNODEMATCHPOLICY
Description	Specifies how node resources should be selected and allocated to the job.
Example	`> qsub -l nodes=2 -W x=nmatchpolicy:exactnode bw.cmd` *Job should use the `EXACTNODE` `JOBNODEMATCHPOLICY`.*

NODESET	
Format	`<SETTYPE>:<SETATTR>[:<SETLIST>]`
Description	Specifies nodeset constraints for job resource allocation. See 7.3 Node Sets for more information.

NODESET	
Example	`> qsub -l nodeset=ONEOF:FEATURE:fastos:hiprio:bigmem bw.cmd`

NODESETCOUNT	
Format	`<INTEGER>`
Description	Specifies how many node sets a job uses.
Example	`> msub -l nodesetcount=2`

NODESETDELAY	
Format	`[[[DD:]HH:]MM:]SS`
Description	Causes Moab to attempt to span a job evenly across nodesets unless doing so delays the job beyond the requested `NODESETDELAY`.
Example	`> qsub -l nodesetdelay=300,walltime=16000 bw.cmd`

NODESETISOPTIONAL	
Format	`<BOOLEAN>`
Description	Specifies whether the nodeset constraint is optional. See 7.3 Node Sets for more information. Requires SCHEDCFG[] FLAGS=allowperjobnodesetisoptional.
Example	`> msub -l nodesetisoptional=true bw.cmd`

OPSYS	
Format	`<OperatingSystem>`
Description	The job's required operating system.

OPSYS	
Example	`> qsub -l nodes=1,opsys=rh73 chem92.cmd`

PARTITION	
Format	`<STRING>[:<STRING>]...`
Description	The partition (or partitions) where the job must run. The job must have access to this partition based on system wide or credential based partition access lists.
Example	`> qsub -l nodes=1,partition=math:geology` *The job must only run in the `math` partition or the `geology` partition.*

PMEM	
Format	`<INTEGER>`
Description	The maximum amount of physical memory used by any single process of the job.
Example	`> msub -l nodes=4:ppn=2,pmem=1024mb` *The job must have `4` compute nodes with `2` processors per node, and each process of the job is limited to 1024 MB of physical memory.*

PREF	
Format	`[{feature|variable}:]<STRING>[:<STRING>]...` If feature or variable are not specified, then feature is assumed.
Description	Specifies which node features are preferred by the job and should be allocated if available. If preferred node criteria are specified, Moab favors the allocation of matching resources but is not bound to only consider these resources.

PREF	
	Preferences are not honored unless the node allocation policy is set to PRIORITY and the PREF priority component is set within the node's PRIORITYF attribute.
Example	`> qsub -l nodes=1,pref=bigmem` *The job can run on any nodes but prefers to allocate nodes with the `bigmem` feature.*

PROCS	
Format	`<INTEGER>`
Description	Requests a specific amount of processors for the job. Instead of users trying to determine the amount of nodes they need, they can instead decide how many processors they need and Moab will automatically request the appropriate amount of nodes from the resource manager. This also works with feature requests, such as `procs=12[:feature1[:feature2[-]]]`. Using this resource request overrides any other processor or node related request, such as `nodes=4`.
Example	`> msub -l procs=32 myjob.pl` *Moab will request as many nodes as is necessary to meet the 32-processor requirement for the job.*

PROLOGUE	
Format	`<STRING>`
Description	Specifies a user owned prologue script, which will be run after the system prologue and `prologue.user` scripts at the beginning of a job. The syntax is `prologue=<file>`. The file can be designated with an absolute or relative path. This parameter works only with Torque.

PROLOGUE	
Example	`> msub -l prologue=prologue_script.sh job.s`

PVMEM	
Format	<INTEGER>
Description	Specify the maximum amount of virtual memory used by any single process in the job.
Example	`> msub -l nodes=4:ppn=2,pvmem=1024mb` *The job must have 4 compute nodes with 2 processors per node, and each process of the job is limited to 1024 MB of virtual memory.*

QoS	
Format	<STRING>
Description	Requests the specified QoS for the job.
Example	`> qsub -l walltime=1000,qos=highprio biojob.cmd`

QUEUEJOB	
Format	<BOOLEAN>
Default	TRUE
Description	Indicates whether or not the scheduler should queue the job if resources are not available to run the job immediately.
Example	`> msub -l nodes=1,queuejob=false test.cmd`

REQATTR	
Format	Required node attributes with version number support: `reqattr=[<must|must not|should|should not>]:<ATTRIBUTE>[{>=|>|<=|<|=}<VERSION>]`
Description	Indicates required node attributes. Values can include letters, numbers, dashes, underscores, periods, and spaces. You can choose one of four requirement types for each node attribute you request: • `must` – The node on which this job runs must include the attribute at the value specified. If no node matches this requirement, Moab will not schedule the job. • `must not` – The node on which this job runs must not include the attribute at the value specified. If no node matches this requirement, Moab will not schedule the job. • `should` – If possible, the node on which this job runs should include the attribute at the value specified. If no node matches this requirement, Moab selects a node without it. • `should not` – If possible, the node on which this job runs should not include the attribute at the value specified. If no node matches this requirement, Moab selects a node without it. If you do not specify a requirement type, Moab assumes 'must.' For information about using reqattr to request dynamic features, see the section Configuring Dynamic Features in Torque and Moab below.
Example	`> qsub -l reqattr=matlab=7.1 testj.txt`

RESFAILPOLICY	
Format	One of `CANCEL`, `HOLD`, `IGNORE`, `NOTIFY`, or `REQUEUE`
Description	The action to take on an executing job if one or more allocated nodes fail. This setting overrides the global value specified with the NODEALLOCRESFAILUREPOLICY parameter.
Example	`> msub -l resfailpolicy=ignore` *For this particular job, ignore node failures.*

RMTYPE	
Format	`<STRING>`
Description	One of the resource manager types currently available within the cluster or grid. Typically, this is `PBS`.
Example	`> msub -l rmtype=pbs` Only run job on a PBS destination resource manager.

SIGNAL	
Format	`<INTEGER>[@<OFFSET>]`
Description	The pretermination signal to be sent to a job prior to it reaching its walltime limit or being terminated by Moab. The optional offset value specifies how long before job termination the signal should be sent. By default, the pretermination signal is sent one minute before a job is terminated. Setting an offset value on a job-end *event* is not supported.
Example	`> msub -l signal=32@120 bio45.cmd`

SPRIORITY	
Format	`<INTEGER>`
Default	0
Description	Allows Moab Admins to set a system priority on a job (similar to setspri). This only works if the job submitter is an admin.
Example	`> qsub -l nodes=16,spriority=100 job.cmd`

TEMPLATE	
Format	`<STRING>`

TEMPLATE	
Description	Specifies a job template to be used as a set template. The requested template must have `SELECT=TRUE` (see Chapter 22: Job Templates).
Example	`> msub -l walltime=1000,nodes=16,template=biojob job.cmd`

TERMTIME	
Format	`<TIMESPEC>`
Default	`0`
Description	The time at which Moab should cancel a queued or active job (see 9.9 Job Deadlines).
Example	`> msub -l nodes=10,walltime=600,termtime=12:00_Jun/14 job.cmd`

TPN	
Format	`<INTEGER>[+]`
Default	`0`
Description	Tasks per node allowed on allocated hosts. If the plus (+) character is specified, the tasks per node value is interpreted as a minimum tasks per node constraint; otherwise it is interpreted as an exact tasks per node constraint. **Differences between TPN and PPN**: There are two key differences between the following: (A) `qsub -l nodes=12:ppn=3` and (B) `qsub -l nodes=12,tpn=3`. The first difference is that `ppn` is interpreted as the *minimum* required tasks per node while `tpn` defaults to exact tasks per node; case (B) executes the job with exactly `3` tasks on each allocated node while case (A) executes the job with at least `3` tasks on each allocated node-`nodeA:4,nodeB:3,nodeC:5` The second major difference is that the line, `nodes=X:ppn=Y` actually requests X*Y tasks, whereas `nodes=X,tpn=Y` requests only X tasks. **TPN with Torque as a Resource Manager**: Moab interprets nodes loosely as procs. Torque interprets nodes as the

TPN	
	number of nodes from the actual number of nodes that you have in your nodes file, not your total number of procs. This means that if Torque is your resource manager and you specify `msub -l nodes=16:tpn=8` but do not have 16 nodes, Torque will not run the job. Instead, you should specify `msub -l procs=16:tpn=8`. To resolve the problem long term, you can also set `server resources_available.nodect` to the total number of procs in your system and use `msub -l nodes=16:tpn=8` as you would in a non-Torque Moab environment. See 'resources_available' in the *Torque Resource Manager Administrator Guide* for more information.
Example	`> msub -l nodes=10,walltime=600,tpn=4 job.cmd`

TRIG	
Format	`<TRIGSPEC>`
Description	Adds triggers to the job (see 17.2.1 Creating a Trigger for specific syntax). Job triggers can only be specified if allowed by the QoS flag trigger. See 17.2.6 Enabling Job Triggers for more information.
Example	`> qsub -l trig=etype=start\&atype=exec\&action="/tmp/email.sh job.cmd"`

TRL (Format 1)	
Format	`<INTEGER>[@<INTEGER>][:<INTEGER>[@<INTEGER>]]...`
Default	`0`
Description	Specifies alternative task requests with their optional walltimes. See 18.3 Malleable Jobs.
Example	`> msub -l trl=2@500:4@250:8@125:16@62 job.cmd` or `> qsub -l trl=2:3:4`

TRL (Format 2)	
Format	`<INTEGER>-<INTEGER>`
Default	`0`
Description	Specifies a range of task requests that require the same walltime. See 18.3 Malleable Jobs.
Example	`> msub -l trl=32-64 job.cmd` For optimization purposes Moab does not perform an exhaustive search of all possible values but will at least do the beginning, the end, and 4 equally distributed choices in between.

VAR	
Format	`<ATTR>[:<VALUE>]`
Description	Adds a generic variable or variables to the job.

Example	`> msub -l VAR=testvar1:testvalue1`
	Single variable
	`> msub -l VAR=testvar1:testvalue1+testvar2:testvalue2+testvar3:testvalue3`
	Multiple variables

VC	
Format	`vc=<NAME>`
Description	Submits the job or workflow to a virtual container (VC).
Example	`vc=vc13`

VMEM	
Format	`<INTEGER>`

VMEM	
Description	Specify the maximum amount of virtual memory used by all concurrent processes in the job.
Example	`> msub -l nodes=4:ppn=2,vmem=1024mb` *The job must have 4 compute nodes with 2 processors per node, and the job is limited to 1024 MB of virtual memory.*

11.3.3 Resource Manager Extension Examples

If more than one extension is required in a given job, extensions can be concatenated with a semicolon separator using the format `<ATTR>:<VALUE>[;<ATTR>:<VALUE>]...`

Example 11-1: Job must run on nodes `node1` and `node2` using the QoS `special`.

```
#@comment="HOSTLIST:node1,node2;QOS:special;SID:silverA"
```

The job is also associated with the system ID `silverA` allowing the silver daemon to monitor and control the job.

Example 11-2: Job will have resources allocated subject to network based nodeset constraints.

```
# PBS -W x=\"NODESET:ONEOF:NETWORK;DMEM:64\"
```

Each task will dedicate 64 MB of memory.

Example 11-3: Job will be forced to run within the `john.1` reservation.

```
>  qsub -l nodes=4,walltime=1:00:00 -W x="FLAGS:ADVRES:john.1"
```

11.3.4 Configuring Dynamic Features in Torque and Moab

Used together, the reqattr RM extension and Torque $varattr parameter allow you to create jobs that request resources that may change or disappear. For example, if you wanted a job to request a certain version of Octave but different versions are configured on each node and updated at any time, you can create a script that searches for the feature and version on the nodes at a specified interval. Your Moab job can then retrieve the dynamic node attributes from the latest poll and use them for scheduling.

This functionality is available when you use the Torque `$varattr` parameter to configure a script that regularly retrieves updates on the nodes' feature(s) and the `reqattr` resource manager extension to require a feature with a certain value.

To set up a dynamic feature in Torque and Moab

1. Create a script that pulls the information you need. For instance, the following script pulls the version of Octave on each node and prints it:

   ```
   #!/bin/bash
   # pull the version string for octave and print it for $varattr
   version_str='octave -v | grep version'
   [[ $version_str =~ ([[:digit:]].[[:digit:]].[[:digit:]]) ]]
   echo "octave=${BASH_REMATCH[1]}"
   ```

2. Use the Torque `$varattr` parameter to configure the script. Specify both the number of seconds between each time Torque runs the script and the path to the script. If you set the seconds to `-1`, the script will run just once. You can include arguments if desired. In the following example, the `varattr` parameter specifies that Torque calls the Octave script every 30 seconds:

   ```
   $varattr 30 /usr/local/scripts/octave.sh
   ```

3. Submit your job in Moab, specifying `reqattr` as a resource. In this example, the job requests a node where the octave feature has a value of 3.2.4 (that the node has Octave version 3.2.4 installed):

   ```
   > msub -l reqattr=octave=3.2.4 myJob.sh
   ```

   Your job requests a node with Octave version 3.2.4. Torque passes the most recent (pulled within the last 30 seconds) version of Octave on each node. Moab then schedules the job on a node that currently has Octave 3.2.4.

Related Topics

- 11.1 Resource Manager
- $varattr in the *Torque Resource Manager Administrator Guide*

11.4 Adding New Resource Manager Interfaces

Moab is designed to interface with multiple resource management systems. Some of these interact through a resource manager specific interface (Torque), while others interact through generalized interfaces such as SSS or Wiki (see Appendix N: Moab Resource Manager Language Interface). For most resource managers, either route is possible depending on where it is easiest to focus development effort. Use of Wiki generally requires modifications on the resource manager side, while creation of a new resource manager specific Moab interface would require more changes to Moab modules.

Regardless of the interface approach selected, adding support for a new resource manager is typically a straightforward process for about 95% of all supported features. The final 5% of features usually requires a bit more effort as each resource manager has a number of distinct concepts that must be addressed.

In this section:

11.4.1 Resource Manager Specific Interfaces

If you require tighter integration and need additional instruction, see 11.5 Managing Resources Directly with the Native Resource Manager Interface. If you want consultation on support for a new resource manager type, contact Adaptive Computing.

11.4.2 Wiki Interface

The Wiki interface is already defined as a resource manager type, so no modifications are required within Moab. Additionally, no resource manager specific library or header file is required. However, within the resource manager, internal job and node objects and attributes must be manipulated and placed within Wiki based interface concepts as defined in Appendix N: Moab Resource Manager Language Interface. Additionally, resource manager parameters must be created to allow a site to configure this interface appropriately.

11.4.3 SSS Interface

The SSS interface is an XML based generalized resource manager interface. It provides an extensible, scalable, and secure method of querying and modifying general workload and resource information.

Related Topics

- 11.5.7 Creating New Tools to Manage the Cluster

11.5 Managing Resources Directly with the Native Resource Manager Interface

In this section:

11.5.1 Native Resource Manager Interface Overview

The Native Resource Manager interface enables you to augment or even fully replace a resource manager for managing resources. In some situations, the full capabilities of the resource manager are not needed and a lower cost or lower overhead alternative is preferred. In other cases, the nature of the environment might make use of a resource manager impossible due to lack of support. Still, in other situations it is desirable to provide information about additional resource attributes, constraints, or state from alternative sources.

In any case, Moab provides the ability to directly query and manage resources along side of or without the use of a resource manager. This interface can also be used to launch, cancel, and otherwise manage jobs. This interface offers several advantages including the following:

- No cost associated with purchasing a resource manager
- No effort required to install or configure the resource manager
- Ability to support abstract resources
- Ability to support abstract jobs
- Ability to integrate node availability information from multiple sources
- Ability to augment node configuration and utilization information provided by a resource manager

However, this interface might also have some drawbacks:

- No support for standard job submission languages
- Limited default configured and utilized resource tracking (additional resource tracking available with additional effort)

At a high level, this interface works by launching threaded calls to perform standard resource manager activities such as managing resources and jobs. The desired calls are configured within Moab and used whenever an action or updated information is required.

11.5.2 Configuring the Native Resource Manager Interface

Using this interface consists of defining the interface type and location. As mentioned earlier, a single object can be fully defined by multiple interfaces simultaneously with each interface updating a particular aspect of the object.

In this topic:

11.5.2.A Configuring the Resource Manager

The Native Resource Manager must be configured using the RMCFG parameter. To specify the Native Resource Manager interface, the `TYPE` attribute must be set to `NATIVE`.

```
RMCFG[local] TYPE=NATIVE
RMCFG[local] CLUSTERQUERYURL=exec:///tmp/query.sh
```

11.5.2.B Reporting Resources

To indicate the source of the resource information, the `CLUSTERQUERYURL` attribute of the `RMCFG` parameter should be specified. This attribute is specified as a URL where the protocols `EXEC` and `FILE` are allowed. If a protocol is not specified, the protocol `EXEC` is assumed.

Format	Description
EXEC	Execute the script specified by the URL path. Use the script stdout as data.
FILE	Load the file specified by the URL path. Use the file contents as data.

Moab considers a Native RM script to have failed if it returns with a non-zero exit code or if the CHILDSTDERRCHECK parameter is set and its appropriate conditions are met. In addition, the Native RM script associated with a job submit URL will be considered as having failed if its standard output stream contains the text ERROR.

This simple example queries a file on the server for information about every node in the cluster. This differs from Moab remotely querying the status of each node individually.

```
RMCFG[local]     TYPE=NATIVE
RMCFG[local]     CLUSTERQUERYURL=file:///tmp/query.txt
```

11.5.3 Generating Flat Cluster Query Data

If the EXEC or FILE protocol is specified in the CLUSTERQUERYURL attribute, the data should provide flat text strings indicating the state and attributes of the node. The format follows the Moab Resource Manager Language Data Format where attributes are delimited by white space rather than ';' (see N.2.1 Query Resources Data Format).

Describes any set of node attributes with format: <NAME><ATTR>=<VAL> [<ATTR>=<VAL>]...

<NAME>	Name of node
<ATTR>	Node attribute
<VAL>	Value of node attribute

```
n17 CPROC=4 AMEMORY=100980 STATE=idle
```

11.5.4 Interfacing with FlexNet (Formerly FLEXlm)

Moab can interface with FlexNet to provide scheduling based on License Management availability. Informing Moab of license dependencies can reduce the number of costly licenses required by your cluster by allowing Moab to intelligently schedule around license limitations.

Provided with Moab in the tools directory is a Perl script, license.mon.flexLM.pl. This script queries a FlexNet license server and gathers data about available licenses. This script then formats this data for Moab to read through a native interface. This script can easily be used by any site to help facilitate FlexNet integration—the only modification necessary to the script is setting the @FLEXlmCmd to specify the local command to query FlexNet. To make this change, edit license.mon.flexLM.pl and, near the top of the file, look for the line:

```
my @FLEXlmCmd = ("SETME");
```

Set the @FLEXlmCmd to the appropriate value for your system to query a license server and license file (if applicable). If lmutil is not in the PATH variable, specify its full path. Using the *lmutil -a* argument will cause it to report all licenses. The -c option is used to specify a license file on a remote server.

```
<path_to_lmstat>/lmstat -c port@host -a
my @FLEXlmCmd = ("<path_to_lmstat>/lmstat -c port@host -a");
```

The @ specifying the port @ servername must be escaped.

To test this script, run it manually. If working correctly, it will produce output similar to the following:

```
> ./license.mon.flexLM.pl
GLOBAL UPDATETIME=1104688300 STATE=idle ARES=autoCAD:130,idl_mpeg:160
CRES=autoCAD:200,idl_mpeg:330
```

If the output looks incorrect, set the $LOGLEVEL variable inside of `license.mon.flexLM.pl`, run it again, and address the reported failure.

Once the license interface script is properly configured, the next step is to add a *license* Native Resource Manager to Moab via the `moab.cfg` file:

```
RMCFG[FLEXlm]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM.pl
...
```

Once this change is made, restart Moab. The command mdiag -R can be used to verify that the resource manager is properly configured and is in the state `Active`. Detailed information regarding configured and utilized licenses can be viewed by issuing the mdiag -n. Floating licenses (non-node-locked) will be reported as belonging to the GLOBAL node.

Due to the inherent conflict with the plus sign (+), the provided license manager script replaces occurrences of the plus sign in license names with the underscore symbol (_). This replacement requires that licenses with a plus sign in their names be requested with an underscore in place of any plus signs.

Interfacing to Multiple License Managers Simultaneously

If multiple license managers are used within a cluster, Moab can interface to each of them to obtain the needed license information. In the case of FlexNet, this can be done by making one copy of the `license.mon.flexLM.pl` script for each license manager and configuring each copy to point to a different license manager. Then, within Moab, create one Native Resource Manager interface for each license manager and point it to the corresponding script as in the following example:

```
RMCFG[FLEXlm1]   TYPE=NATIVE RESOURCETYPE=LICENSE
```

```
RMCFG[FLEXlm1]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM1.pl
RMCFG[FLEXlm2]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm2]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM2.pl
RMCFG[FLEXlm3]   TYPE=NATIVE RESOURCETYPE=LICENSE
RMCFG[FLEXlm3]   CLUSTERQUERYURL=exec://$TOOLSDIR/flexlm/license.mon.flexLM3.pl
...
```

For an overview of license management, including job submission syntax, see 11.7 License Management.

It may be necessary to increase the default limit, MMAX_GRES. See Appendix D: Adjusting Default Limits for more implementation details.

11.5.5 Interfacing to Nagios

Moab can interface with Nagios to provide scheduling based on network hosts and services availability.

Nagios installation and configuration documentation can be found at Nagios.org.

Provided with Moab in the tools directory is a Perl script, node.query.nagios.pl. This script reads the Nagios `status.dat` file and gathers data about network hosts and services. This script then formats data for Moab to read through a native interface. This script can be used by any site to help facilitate Nagios integration. To customize the data that will be formatted for Moab, make the changes in this script.

You may need to customize the associated configuration file in the etc directory, `config.nagios.pl`. The statusFile line in this script tells Moab where the Nagios `status.dat` file is located. Make sure that the path name specified is correct for your site. Note that the interval that Nagios updates the Nagios status.dat file is specified in the Nagios `nagios.cfg` file. Refer to Nagios documentation for more information.

To make these changes, familiarize yourself with the format of the Nagios `status.dat` file and make the appropriate additions to the script to include the desired Moab Resource Manager language (formerly WIKI) Interface attributes in the Moab output.

To test this script, run it manually. If working correctly, it will produce output similar to the following:

```
> ./node.query.nagios.pl
gateway STATE=Running
localhost STATE=Running CPULOAD=1.22 ADISK=75332
```

Once the Nagios interface script is properly configured, the next step is to add a Nagios Native Resource Manager to Moab via the `moab.cfg` file:

```
RMCFG[nagios] TYPE=NATIVE
RMCFG[nagios] CLUSTERQUERYURL=exec://$TOOLSDIR/node.query.nagios.pl
...
```

Once this change is made, restart Moab. The command mdiag -R can be used to verify that the resource manager is properly configured and is in the state `Active`. Detailed information regarding configured Nagios node information can be viewed by issuing the mdiag -n.

```
> mdiag -n -v
compute node summary
Name                    State   Procs      Memory         Disk          Swap
Speed   Opsys   Arch Par   Load Rsv Classes                        Network
           Features
gateway               Running    0:0         0:0           0:0           0:0
1.00       -      - dav   0.00   0 -                             -
          -
  WARNING:  node 'gateway' is busy/running but not assigned to an active job
  WARNING:  node 'gateway' has no configured processors
localhost             Running    0:0         0:0       75343:75347       0:0
1.00       -      - dav   0.48   0 -                             -
          -
  WARNING:  node 'localhost' is busy/running but not assigned to an active job
  WARNING:  node 'localhost' has no configured processors
-----                   ---     3:8      1956:1956    75345:75349    5309:6273
Total Nodes: 2  (Active: 2  Idle: 0  Down: 0)
```

11.5.6 Configuring Resource Types

Native Resource Managers can also perform special tasks when they are given a specific resource type. These types are specified using the RESOURCETYPE attribute of the RMCFG parameter.

Type	Description
COMPUTE	Normal compute resources (no special handling).
FS	File system resource manager (see 11.6 Utilizing Multiple Resource Managers for an example).
LICENSE	Software license manager (see 11.5.4 Interfacing with FlexNet (Formerly FLEXlm) and 11.7 License Management). If LICENSE is specified, *all* generic resources reported by the resource manager will be marked as a license; causing them to be tracked by the accounting manager under the Licenses property. See the property Licenses for more information on the Licenses property.
NETWORK	Network resource manager.

Type	Description
PROV	Provisioning resource manager. This is the resource manager that Moab uses to modify the OS of a node and to power a node on or off.

11.5.7 Creating New Tools to Manage the Cluster

Using the scripts found in the `$TOOLSDIR` (`$INSTDIR/tools`) directory as a template, new tools can be quickly created to monitor or manage most any resource. Each tool should be associated with a particular resource manager service and specified using one of the following resource manager URL attributes.

CLUSTERQUERYURL	
Description	Queries resource state, configuration, and utilization information for compute nodes, networks, storage systems, software licenses, and other resources. For more information, see the attribute CLUSTERQUERYURL.
Output	Node status and configuration for one or more nodes. See N.2.1 Query Resources Data Format.
Example	`RMCFG[v-stor] CLUSTERQUERYURL=exec://$HOME/storquery.pl` *Moab will execute the `storquery.pl` script to obtain information about 'v-stor' resources.*

JOBCANCELURL	
Description	Cancels a job.
Input	`<JOBID>`
Example	`RMCFG[v-stor] JOBCANCELURL=exec://$HOME/cancel.pl` *Moab will execute the `cancel.pl` script to cancel jobs.*

JOBMODIFYURL	
Description	Modifies a job or application. For more information, see the attribute JOBMODIFYURL.

JOBMODIFYURL	
Input	`[-j <JOBEXPR>] [--s[et]|--u[nset]|--c[lear]|--i[ncrement]|--d[ecrement]] <ATTR>[=<VALUE>] [<ATTR>[=<VALUE>]]...`
Example	`RMCFG[v-stor] JOBMODIFYURL=exec://$HOME/jobmodify.pl` *Moab will execute the* `jobmodify.pl` *script to modify the specified job.*

JOBREQUEUEURL	
Description	Requeues a job.
Input	`<JOBID>`
Example	`RMCFG[v-stor] JOBREQUEUEURL=exec://$HOME/requeue.pl` *Moab will execute the* `requeue.pl` *script to requeue jobs.*

JOBRESUMEURL	
Description	Resumes a suspended job or application.
Input	`<JOBID>`
Example	`RMCFG[v-stor] JOBRESUMEURL=exec://$HOME/jobresume.pl` *Moab will execute the* `jobresume.pl` *script to resume suspended jobs.*

JOBSTARTURL	
Description	Launches a job or application on a specified set of resources.
Input	`<JOBID><TASKLIST><USERNAME> [ARCH=<ARCH>] [OS=<OPSYS>] [IDATA=<STAGEINFILEPATH>[,<STAGEINFILEPATH>]...] [EXEC=<EXECUTABLEPATH>]`
Example	`RMCFG[v-stor] JOBSTARTURL=exec://$HOME/jobstart.pl` *Moab will execute the* `jobstart.pl` *script to execute jobs.*

JOBSUBMITURL	
Description	Submits a job to the resource manager, but it does not execute the job. The job executes when the `JOBSTARTURL` is called.
Input	`[ACCOUNT=<ACCOUNT>] [ERROR=<ERROR>] [GATTR=<GATTR>] [GNAME=<GNAME>] [GRES=<GRES>:<Value>[,<GRES>:<Value>]*] [HOSTLIST=<HOSTLIST>] [INPUT=<INPUT>] [IWD=<IWD>] [NAME=<NAME>] [OUTPUT=<OUTPUT>] [RCLASS=<RCLASS>] [REQUEST=<REQUEST>] [RFEATURES=<RFEATURES>] [RMFLAGS=<RMFLAGS>] [SHELL=<SHELL>] [TASKLIST=<TASKLIST>] [TASKS=<TASKS>] [TEMPLATE=<TEMPLATE>] [UNAME=<UNAME>] [VARIABLE=<VARIABLE>] [WCLIMIT=<WCLIMIT>] [ARGS=<Value>[<Value>]*]` ARGS must be the last submitted attribute because there can be multiple space-separated values for ARGS.
Example	`RMCFG[v-stor] JOBSUBMITURL=exec://$HOME/jobsubmit.pl` *Moab submits the job to the `jobsubmit.pl` script for future job execution.*

JOBSUSPENDURL	
Description	Suspends in memory an active job or application.
Input	`<JOBID>`
Example	`RMCFG[v-stor] JOBSUSPENDURL=exec://$HOME/jobsuspend.pl` *Moab will execute the `jobsuspend.pl` script to suspend active jobs.*

NODEMODIFYURL	
Description	Provide method to dynamically modify/provision compute resources including operating system, applications, queues, node features, power states, etc.

NODEMODIFYURL	
	Moab blocks scheduling when invoking the script defined by NODEMODIFYURL. If your NODEMODIFYURL script is a long running process (i.e., > 30 seconds or, if defined, longer than the RMCFG TIMEOUT setting), then you must background the process and return quickly.
Input	`<NODEID>[,<NODEID>] [--force] {--set <ATTR>=<VAL>|--clear <ATTR>}` ATTR is one of the node attributes listed in N.2.1 Query Resources Data Format.
Example	`RMCFG[warewulf] NODEMODIFYURL=exec://$HOME/provision.pl` *Moab will reprovision compute nodes using the `provision.pl` script.*

NODEPOWERURL	
Description	Allows Moab to issue IPMI power commands.
Input	`<NODEID>[,<NODEID>] ON | OFF`
Example	`RMCFG[node17rm] NODEPOWERURL=exec://$TOOLSDIR/ipmi.power.pl` *Moab will issue a power command contained in the `ipmi.power.pl` script.*

SYSTEMMODIFYURL	
Description	Provide method to dynamically modify aspects of the compute environment that are directly associated with cluster resources. For more information, see the attribute SYSTEMMODIFYURL.

SYSTEMQUERYURL	
Description	Provide method to dynamically query aspects of the compute environment that are directly associated with cluster resources. For more information, see the attribute SYSTEMQUERYURL.
Input	Default `<ATTR>`

SYSTEMQUERYURL	
	ATTR is one of `images`
Output	`<STRING>`
Example	`RMCFG[warewulf] SYSTEMQUERYURL=exec://$HOME/checkimage.pl` *Moab will load the list of images available from warewulf using the* `checkimage.pl` *script.*

WORKLOADQUERYURL	
Description:	Provide method to dynamically query the system workload (jobs, services, etc.) of the compute environment that are associated with managed resources. Job/workload information should be reported back from the URL (script, file, web service, etc.) using the Moab Resource Manager Language Data Format (formerly WIKI). For more information, see the attribute WORKLOADQUERYURL.
Output:	`<STRING>`
Example	`RMCFG[xt] WORKLOADQUERYURL=exec://$HOME/job.query.xt3.pl` *Moab will load job/workload information by executing the* `job.query.xt3.pl` *script.*

Related Topics

- 3.7.14 mdiag -R (command to evaluate resource managers)
- 11.7 License Management
- N.2 Moab Resource Manager Language Data Format

11.6 Utilizing Multiple Resource Managers

In this section:

11.6.1 Multi-Resource Manager Overview

In many instances, a site may have certain resources controlled by different resource managers. For example, a site may use a particular resource manager for licensing software for jobs, another resource manager for managing file systems, another resource manager for job control, and another for node monitoring. Moab can be configured to communicate with each of these resource managers, gathering all their data and incorporating such into scheduling decisions. With a more distributed approach to resource handling, failures are more contained and scheduling decisions can be more intelligent.

11.6.2 Configuring Multiple Independent Resource Manager Partitions

Moab must know how to communicate with each resource manager. In most instances, this is simply done by configuring a query command.

11.6.3 Migrating Jobs between Resource Managers

With multi-resource manager support, a job can be submitted either to a local resource manager queue or to the Moab global queue. In most cases, submitting a job to a resource manager queue constrains the job to only run within the resources controlled by that resource manager. However, if the job is submitted to the Moab global queue, it can use resources of any active resource manager. This is accomplished through job translation and staging.

When Moab evaluates resource availability, it determines the cost in terms of both data and job staging. If staging a job's executable or input data requires a significant amount of time, Moab integrates data and compute resource availability to determine a job's earliest potential start time on a per resource manager basis and makes an optimal scheduling decision accordingly. If the optimal decision requires a data stage operation, Moab reserves the required compute resources, stages the data, and then starts the job when the required data and compute resources are available.

11.6.4 Aggregating Information into a Cohesive Node View

Using the native interface, Moab can actually perform most of these functions without the need for an external resource manager. First, configure the Native Resource Managers:

```
RMCFG[base]     TYPE=PBS
RMCFG[network]  TYPE=NATIVE
RMCFG[network]  CLUSTERQUERYURL=/tmp/network.sh
RMCFG[fs]       TYPE=NATIVE
RMCFG[fs]       CLUSTERQUERYURL=/tmp/fs.sh
```

The network script can be as simple as the following:

```
> _RX=`/sbin/ifconfig eth0 | grep "RX by" | cut -d: -f2 | cut -d' ' -f1`; \
> _TX=`/sbin/ifconfig eth0 | grep "TX by" | cut -d: -f3 | cut -d' ' -f1`; \
> echo `hostname` GMETRIC[netusage]=`echo "$_RX + $_TX" | bc`;
```

The preceding script will output something such as the following:

```
node01 GMETRIC[netusage]=10928374
```

Moab grabs information from each resource manager and includes its data in the final view of the node:

```
> checknode node01
node node01
State:   Running  (in current state for 00:00:20)
Configured Resources: PROCS: 2  MEM: 949M  SWAP: 2000M  disk: 1000000
Utilized   Resources: SWAP: 9M
Dedicated  Resources: PROCS: 1  disk: 1000
Opsys:      Linux-2.6.5-1.358  Arch:      linux
Speed:      1.00  CPULoad:       0.320
Location:   Partition: DEFAULT  Rack/Slot:  NA
Network Load: 464.11 b/s
Network:    DEFAULT
Features:   fast
Classes:    [batch 1:2][serial 2:2]
Total Time: 00:30:39  Up: 00:30:39 (100.00%)  Active: 00:09:57 (32.46%)
Reservations:
  Job '5452'(x1)  -00:00:20 -> 00:09:40 (00:10:00)
JobList:  5452
```

Notice that the Network Load is now being reported along with disk usage.

Example File System Utilization Tracker (per user)

The following configuration can be used to track file system usage on a per user basis:

```
.....
RMCFG[file]     TYPE=NATIVE
RMCFG[file]     RESOURCETYPE=FS
RMCFG[file]     CLUSTERQUERYURL=/tmp/fs.pl
.....
```

Assuming that `/tmp/fs.pl` outputs something of the following format:

```
DEFAULT STATE=idle AFS=<fs id="user1" size="789456"></fs><fs
id="user2" size="123456"></fs>
```

This will track disk usage for users *user1* and *user2* every 24 hours.

11.7 License Management

In this section:

11.7.1 License Management Overview

Software license management is typically enabled in one of two models: node-locked and floating. Under a node-locked license, use of a given application is constrained to certain hosts. For example, `node013` can support up to two simultaneous jobs accessing application matlab. In a floating license model, a limited number of software licenses are made available cluster wide, and these licenses can be used on any combination of compute hosts. In each case, these licenses are consumable and application access is denied once they are gone.

Moab supports both node-locked and floating license models and even allows mixing the two models simultaneously. Moab monitors license usage and only launches an application when required software license availability is guaranteed. In addition, Moab also reserves licenses in conjunction with future jobs to ensure these jobs can run at the appropriate time.

By default, Moab supports up to 128 independent license types.

Moab license recognition is case insensitive. This means that two licenses with the same spelling and different capitalization are still recognized as the same license. When this occurs, Moab considers the license invalid.

11.7.2 Controlling and Monitoring License Availability

Moab can use one of three methods to determine license availability. These methods include locally specifying consumable generic resources, obtaining consumable generic resource information from the resource manager, and interfacing directly with a license manager.

In this topic:

11.7.2.A Local Consumable Resources

Both node-locked and floating licenses can be locally specified within Moab using the NODECFG parameter. In all cases, this is accomplished by associating the license with a node using the GRES (or generic resource) attribute. If floating, the total cluster-wide license count should be associated with the GLOBAL node. If node-locked, the per node license count should be associated with each compute host (or globally using the `DEFAULT` node). For example, if you have two node-locked licenses for application `EvalA` and six floating licenses for application `EvalB`, the following configuration could be used:

```
NODECFG[node001]  GRES=EvalA:2
NODECFG[node002]  GRES=EvalA:2
NODECFG[GLOBAL]   GRES=EvalB:6
...
```

If you are using an accounting manager and want to distinguish certain generic resources as 'Licenses', you can use the GRESCFG[] LICENSE=TRUE parameter in the Moab server configuration. See the parameter GRESCFG[<GRES>] for more information.

11.7.2.B Resource Manager Based Consumable Resources

Some resource managers support the ability to define and track generic resource usage at a per node level. In such cases, support for node-locked licenses can be enabled by specifying this information within the resource manager. Moab automatically detects and schedules these resources. For example, in the case of Torque, this can be accomplished by adding generic resource specification lines to the MOM configuration file.

11.7.2.C Interfacing to an External License Manager

Moab can also obtain live software license information from a running license manager. Direct interfaces to supported license managers such as FlexNet (formerly FLEXlm) can be created using the Native Resource Manager feature. A complete example on interfacing to an external license manager is provided in the FlexNet section of the Native Resource Manager overview.

Interfacing to Multiple License Managers

Moab can interface to multiple external license managers simultaneously simply by defining additional Native Resource Manager interfaces. See 11.5.1 Native Resource Manager Interface Overview for more information.

11.7.3 Requesting Licenses within Jobs

Requesting use of software licenses within jobs is typically done in one of two ways. In most cases, the Native Resource Manager job submission language provides a direct method of license specification; for example, in the case of Torque, the software argument could be specified using the format `<SOFTWARE_NAME>[+<LICENSE_COUNT>]` as in the following example:

```
> qsub -l nodes=2,software=blast cmdscript.txt
```

Known issues have been reported using 'software'. The '-l software' syntax is scheduled to be deprecated. We recommend using 'gres' instead. For example:

```
> qsub -l nodes=2,gres=blast cmdscript.txt
```

The license count is a job total, not a per task total, and the license count value defaults to 1.

An alternative to direct specification is the use of the Moab resource manager extensions. With these extensions, licenses can be requested as generic resources, using the GRES attribute. The job in the preceding example could also be requested using the following syntax:

```
> qsub -l nodes=2 -W x=GRES:blast cmdscript.txt
```

In each case, Moab automatically determines if the software licenses are node-locked or floating and applies resource requirements accordingly.

If a job requires multiple software licenses, whether of the same or different types, a user will use the following syntax:

```
> qsub -l nodes=2 -W x=GRES:blast+2 cmdscript.txt   # two 'blast' licenses required
> qsub -l nodes=2 -W x=GRES:blast+2%bkeep+3 cmdscript.txt   # two 'blast' and three
'bkeep' licenses are required
```

Related Topics

- 11.5.4 Interfacing with FlexNet (Formerly FLEXlm)
- 6.1 Advance Reservations
- Chapter 23: Moab Workload Manager for Grids

11.8 Resource Provisioning

In this section:

11.8.1 Resource Provisioning Overview

When processing a resource request, Moab attempts to match the request to an existing available resource. However, if the scheduler determines that the resource is not available or will not be available due to load or policy for an appreciable amount of time, it can select a resource to modify to meet the needs of the current requests. This process of modifying resources to meet existing needs is called provisioning.

Moab evaluates the costs of making the provisioning change in terms of time and other resources consumed before making the decision. Only if the benefits outweigh the costs will the scheduler initiate the change required to support the current workload.

When Moab provisions an environment, it provisions (on a per node basis) the OS and its associated libraries, applications, etc. as a single environment.

Preemption (requeuing) does not work with dynamic provisioning.

11.8.2 Configuring Provisioning

Enabling provisioning consists of configuring an interface to a provisioning manager, specifying which nodes can take advantage of this service, and what the estimated cost and duration of each change will be. This interface can be used to contact provisioning software such as xCat or HP's Server Automation tool. Additionally, locally developed systems can be interfaced via a script or web service.

Related Topics

- 11.5 Managing Resources Directly with the Native Resource Manager Interface
- Appendix O: Resource Manager Integration

11.9 Managing Networks

In this section:

11.9.1 Network Management

Network resources can be tightly integrated with the rest of a compute cluster using the Moab multi-resource manager management interface. This interface has the following capabilities:

- Dynamic per job and per partition VLAN creation and management
- Monitoring and reporting of network health and failure events
- Monitoring and reporting of network load
- Creation of subnets with guaranteed performance criteria
- Automated workload-aware configuration and router maintenance
- Intelligent network-aware scheduling algorithms

11.9.2 Dynamic VLAN Creation

Most sites using dynamic VLANs operate under the following assumptions:

- Each compute node has access to two or more networks, one of which is the compute network, and another which is the admin network.
- Each compute node can only access other compute nodes via the compute network.

- Each compute node can only communicate with the head node via the administrator network.
- Logins on the head node cannot be requested from a compute node.

In this environment, organizations may choose to have VLANs automatically configured that encapsulate individual jobs. These VLANs essentially disconnect the job from either incoming or outgoing communication with other compute nodes.

In this topic:

11.9.2.A Configuring VLANs

Automated VLAN management can be enabled by setting up a network resource manager that supports dynamic VLAN configuration and a QoS to request this feature. The example configuration highlights this setup:

```
...
RMCFG[cisco] TYPE=NATIVE RESOURCETYPE=NETWORK FLAGS=VLAN
RMCFG[cisco] CLUSTERQUERYURL=exec://$TOOLSDIR/node.query.cisco.pl
RMCFG[cisco] SYSTEMMODIFYURL=exec://$TOOLSDIR/system.modify.cisco.pl
QOSCFG[netsecure] SECURITY=VLAN
```

11.9.2.B Requesting a VLAN

VLANs can be requested on a per job basis directly using the associated resource manager extension or indirectly by requesting a QoS with a VLAN security requirement:

```
> qsub -l nodes=256,walltime=24:00:00,qos=netsecure biojob.cmd
143325.umc.com submitted
```

11.9.3 Network Load and Health Monitoring

Network-level load and health monitoring is enabled by supporting the cluster query action in the network resource manager and specifying the appropriate CLUSTERQUERYURL attribute in the associated resource manager interface. Node (virtual node) query commands (mnodectl, checknode) can be used to view this load and health information that will also be correlated with associated workload and written to persistent accounting records. Network load and health based event information can also be fed into generic events and used to drive appropriate event based triggers.

At present, load and health attributes such as fan speed, temperature, port failures, and various core switch failures can be monitored and reported. Additional failure events are monitored and reported as support is added within the network management system.

11.9.4 Creating a Resource Management Interface for a New Network

Many popular networks are supported using interfaces provided in the Moab `tools` directory. If a required network interface is not available, a new one can be created using the following guidelines.

In this topic:

11.9.4.A General Requirements

In all cases, a network resource manager should respond to a cluster query request by reporting a single node with a node name that will not conflict with any existing compute nodes. This node should report as a minimum the `state` attribute.

11.9.4.B Monitoring Load

Network load is reported to Moab using the generic resource bandwidth. For greatest value, both configured and used bandwidth (in megabytes per second) should be reported as in the following example:

```
force10 state=idle ares=bandwidth:5466 cres=bandwidth:10000
```

11.9.4.C Monitoring Failures

Network warning and failure events can be reported to Moab using the `gevent` metric. If automated responses are enabled, embedded epochtime information should be included.

```
force10 state=idle gevent[checksum]='ECC failure detected on port 13'
```

11.9.4.D Controlling Router State

Router power state can be controlled as a system modify interface is created that supports the commands on, off, and reset.

11.9.4.E Creating VLANs

VLAN creation, management, and reporting is more advanced requiring persistent VLAN ID tracking, global pool creation, and other features. Use of existing routing interface tools as templates is highly advised. VLAN management requires use of both the cluster query interface and the system modify interface.

11.9.5 Per-Job Network Monitoring

It is possible to gather network usage on a per job basis using the Native Interface. When the native interface has been configured to report netin and netout Moab automatically gathers this data through the life of a job and reports total usage statistics upon job completion.

```
...
node99  netin=78658 netout=1256
...
```

This information is visible to users and admins via command-line utilities, the web portal, and the desktop graphical interfaces.

Related Topics

- 11.5 Managing Resources Directly with the Native Resource Manager Interface
- 14.3 Workload Event Format

11.10 Intelligent Platform Management Interface

In this section:

11.10.1 IPMI Overview

The Intelligent Platform Management Interface (IPMI) specification defines a set of common interfaces system admins can use to monitor system health and manage the system. The IPMI interface can monitor temperature and other sensor information, query platform status and power-on/power-off compute nodes. As IPMI operates independently of the node's OS interaction with the node can happen even when powered down. Moab can use IPMI to monitor temperature information, check power status, power-up, power-down, and reboot compute nodes.

11.10.2 Node IPMI Configuration

IPMI must be enabled on each node in the compute cluster. This is usually done either through the node's BIOS or by using a boot CD containing IPMI utilities provided by the manufacturer. With regard to configuring IPMI on the nodes, be sure to enable IPMI-over-LAN and set a common login and password on all the nodes. Additionally, you must set a unique IP address for each node's BMC. Take note of these addresses as you will need them when reviewing the section Creating the IPMI BMC-Node Map File.

11.10.3 Installing IPMItool

IPMItool is an open-source tool used to retrieve sensor information from the IPMI Baseboard Management Controller (BMC) or to send remote chassis power control commands. The IPMItool developer provides Fedora Core binary packages, and a source tarball on the IPMItool download page. Download and install IPMItool on the Moab head node and make sure the `ipmitool` binary is in the current shell PATH.

Proper IPMI setup and IPMItool configuration can be confirmed by issuing the following command on the Moab head node:

```
> ipmitool -I lan -U username -P password -H BMC IP chassis status
```

The output of this command should be similar to the following:

```
System Power         : off
Power Overload       : false
Power Interlock      : inactive
Main Power Fault     : false
```

```
Power Control Fault  : false
Power Restore Policy : previous
Last Power Event     :
Chassis Intrusion    : inactive
Front-Panel Lockout  : inactive
Drive Fault          : false
Cooling/Fan Fault    : false
```

11.10.4 [Optional] Creating the IPMI BMC-Node Map File

Since the BMC can be controlled via LAN, it is possible for the BMC to have its own unique IP address. Since this IP address is separate from the IP address of the node, a simple mapping file is required for Moab to know each node's BMC address. The file is a flat text file and should be stored in the Moab home directory. If a mapping file is needed, specify the name in the `config.ipmi.pl` configuration file in the `etc/` directory. The following is an example of the mapping file:

```
#<NodeID> <BMC IP>
node01  10.10.10.101
node02  10.10.10.102
node03  10.10.10.103
node04  10.10.10.104
node05  10.10.10.105
# NodeID = the name of the nodes returned with "mdiag -n"
# BMC IP = the IP address of the IPMI BMC network interface
```

Note that only the nodes specified in this file are queried for IPMI information. Also note that the mapping file is disabled by default and the nodes that are returned from Moab with *`mdiag -n`* are the ones that are queried for IPMI sensor data.

11.10.5 Configuring the Moab IPMI Tools

The `tools/` subdirectory in the install directory already contains the Perl scripts needed to interface with IPMI. The following is a list of the Perl scripts that should be in the `tools/` directory; confirm these are present and executable:

```
ipmi.mon.pl     # The daemon front-end called by Moab
ipmi.power.pl   # The power control script called by Moab
__mon.ipmi.pl   # The IPMI monitor daemon that updates and caches IPMI data from nodes
```

Next, a few configuration settings need to be adjusted in the `config.ipmi.pl` file found in the `etc` subdirectory. The IPMI-over-LAN username and password need to be set to the values that were set in the section Node IPMI Configuration. Also, the IPMI query daemon's polling interval can be modified by adjusting $pollInterval. This specifies how often the IPMI-enabled nodes are queried to retrieve sensor data.

11.10.6 Configuring Moab

To allow Moab to use the IPMI tools, a Native Resource Manager is configured. To do this, the following lines must be added to `moab.cfg`:

```
...
# IPMI - Node monitor script
RMCFG[ipminative] TYPE=NATIVE CLUSTERQUERYURL=exec://$TOOLSDIR/ipmi.mon.pl
...
```

Next, the following lines can be added to allow Moab to issue IPMI power commands:

```
...
# IPMI - Power on/off/reboot script
RMCFG[ipminative] NODEPOWERURL=exec://$TOOLSDIR/ipmi.power.pl
...
```

Moab can be configured to perform actions based on sensor data. For example, Moab can shut down a compute node if its CPU temperature exceeds 100 degrees Celsius, or it can power down idle compute nodes if workload is low. Generic event thresholds are used to tell Moab to perform certain duties given certain conditions. The following example is a way for Moab to recognize it should power off a compute node if its CPU0 temperature exceeds 100 degrees Celsius:

```
...
# IPMI - Power off compute node if its CPU0 temperature exceeds 100 degrees Celsius.
GEVENTCFG[CPU0_TEMP>100] action=off
...
```

11.10.7 Ensuring Proper Setup

Once the preceding steps have been taken, Moab should be started as normal. The IPMI monitoring daemon should start automatically, which can be confirmed with the following:

```
moab@headnode:~/$ ps aux | grep __mon
moab   11444  0.0  0.3   6204  3172 pts/3    S    10:54   0:00 /usr/bin/perl -w
/opt/moab/tools/_mon.ipmi.pl --start
```

After a few minutes, IPMI data should be retrieved and cached. This can be confirmed with the following command:

```
moab@headnode:~/$ cat spool/ipmicache.gm
node01 GMETRIC[CPU0_TEMP]=49
node01 GMETRIC[CPU1_TEMP]=32
node01 GMETRIC[SYS_TEMP]=31
node01 POWER=ON
```

Finally, issue the following to ensure Moab is grabbing the IPMI data:

```
moab@headnode:~/$ checknode node01
node node01
State:      Idle  (in current state for 00:03:12)
```

```
Configured Resources: PROCS: 1  MEM: 2000M  SWAP: 3952M  DISK: 1M
Utilized   Resources: ---
Dedicated  Resources: ---
Generic Metrics:  CPU0_TEMP=42.00,CPU1_TEMP=30.00,SYS_TEMP=29.00
...
```

Temperature data should be present in the `Generic Metrics` row.

11.11 Resource Manager Translation

In this section:

11.11.1 Translation Overview

Resource manager translation enables end-users to continue to use existing job command scripts and familiar job management and resource query commands. This is accomplished by emulating external commands, routing the underlying queries to Moab, and then formatting the responses in a familiar manner. Using translation, job submission clients, job query clients, job control clients, and resource query clients can be emulated making switching from one resource manager to another transparent and preserving investment in legacy scripts, tools, and experience.

11.11.2 Translation Enablement Steps

To enable translation, you must:

- Edit the Moab tools configuration file.
- Copy, rename, and link the emulation scripts to a shorter, easier-to-use name.

In this topic:

11.11.2.A Configure Translation Tools

Located in the `$MOABHOMEDIR/etc` directory are tools-specific configuration files. For each resource manager that has installed translation tools, edit the Moab tools configuration file in the `etc` directory. For example, do the following:

```
> vi $MOABHOMEDIR/etc/config.moab.pl
# Set the PATH to include directories for moab client commands — mjobctl, etc.
$ENV{PATH} = "/opt/moab/bin:$ENV{PATH}";
```

11.11.2.B Add Translation Tools

In a directory accessible to users, create links to (or copy) the emulation scripts you want your users to use. For example, the emulation script `tools/bjobs.lsf.pl` could be copied to `bin/bjobs`, or, a symbolic link could be created in `bin/bjobs` that points to `tools/bjobs.lsf.pl`.

```
> ln -s tools/bjobs.lsf.pl bin/bjobs
> ln -s tools/bhosts.lsf.pl bin/bhosts
```

Chapter 12: Troubleshooting and System Maintenance

In this chapter:

12.1 Internal Diagnostics/Diagnosing System Behavior and Problems

Moab provides a number of commands for diagnosing system behavior. These diagnostic commands present detailed state information about various aspects of the scheduling problem, summarize performance, and evaluate current operation reporting on any unexpected or potentially erroneous conditions found. Where possible, Moab's diagnostic commands even correct detected problems if desired.

At a high level, the diagnostic commands are organized along functionality and object based delineations. Diagnostic commands exist to help prioritize workload, evaluate fairness, and determine effectiveness of scheduling optimizations. Commands are also available to evaluate reservations reporting state information, potential reservation conflicts, and possible corruption issues. Scheduling is a complicated task. Failures and unexpected conditions can occur as a result of resource failures, job failures, or conflicting policies.

Moab's diagnostics can intelligently organize information to help isolate these failures and allow them to be resolved quickly. Another powerful use of the diagnostic commands is to address the situation where there are no hard failures. In these cases, the jobs, compute nodes, and scheduler are all functioning properly, but the cluster is not behaving exactly as desired. Moab diagnostics can help you determine how the current configuration is performing and how it can be changed to obtain the desired behavior.

In this section:

12.1.1 The mdiag Command

The cornerstone of Moab's diagnostics is the mdiag command. This command provides detailed information about scheduler state and also performs a large number of internal sanity checks presenting problems it finds as warning messages.

Currently, the *mdiag* command provides in-depth analysis of the following objects and subsystems:

Object/Subsystem	mdiag Flag	Use
Account	-a	Shows detailed account configuration information.
Blocked	-b	Indicates why blocked (ineligible) jobs are not allowed to run.
Class	-c	Shows detailed class configuration information.
Config	-C	Shows configuration lines from `moab.cfg` and whether or not they are valid.
FairShare	-f	Shows detailed fairshare configuration information, and current fairshare usage.
Group	-g	Shows detailed group information.
Job	-j	Shows detailed job information. Reports corrupt job attributes, unexpected states, and excessive job failures.
Frame/Rack	-m	Shows detailed frame/rack information.
Node	-n	Shows detailed node information. Reports unexpected node states and resource allocation conditions.
Priority	-p	Shows detailed job priority information including priority

Object/Subsystem	mdiag Flag	Use
		factor contributions to all idle jobs.
QoS	-q	Shows detailed QoS information.
Reservation	-r	Shows detailed reservation information. Reports reservation corruption and unexpected reservation conditions.
Resource Manager	-R	Shows detailed resource manager information. Reports configured and detected state, configuration, performance, and failures of all configured resource manager interfaces.
Standing Reservations	-s	Shows detailed standing reservation information. Reports reservation corruption and unexpected reservation conditions.
Scheduler	-S	Shows detailed scheduler state information. Indicates if scheduler is stopped, reports status of grid interface, and identifies and reports high-level scheduler failures.
Partition	-t	Shows detailed partition information.
User	-u	Shows detailed user information.

12.1.2 Other Diagnostic Commands

Beyond *mdiag*, the checkjob and checknode commands also provide detailed information and sanity checking on individual jobs and nodes respectively. These commands can indicate why a job cannot start, which nodes can be available, and information regarding the recent events impacting current job or nodes state.

12.1.3 Using Moab Logs for Troubleshooting

Moab logging is extremely useful in determining the cause of a problem. Where other systems may be criticized for not providing adequate logging to diagnose a problem, Moab may be criticized for the opposite reason. If the logging level is configured too high, huge volumes of log output might be recorded, potentially obscuring the problems in a flood of data. Intelligent searching combined with the use of the LOGLEVEL and LOGFACILITY parameters can mine out the needed information. Key information associated with various problems is generally marked with the keywords WARNING, ALERT, or ERROR. See 12.2 Logging for more information.

12.1.4 Automating Recovery Actions after a Failure

The RECOVERYACTION parameter of SCHEDCFG can be used to control scheduler action in the case of a catastrophic internal failure. Valid actions include die, ignore, restart, and trap.

Recovery Mode	Description
die	Moab will exit and, if core files are externally enabled, create a core file for analysis (this is the default behavior).
ignore	Moab will ignore the signal and continue processing. This might cause Moab to continue running with corrupt data, which might be dangerous. **Note:** Use this setting with caution.
restart	When a SIGSEGV is received, Moab will relaunch using the current checkpoint file, the original launch environment, and the original command line flags. The receipt of the signal will be logged but Moab will continue scheduling. Because the scheduler is restarted with a new memory image, no corrupt scheduler data should exist. One caution with this mode is that it might mask underlying system failures by allowing Moab to overcome them. If used, the event log should be checked occasionally to determine if failures are being detected.
trap	When a SIGSEGV is received, Moab stays alive but enters diagnostic mode. In this mode, Moab stops scheduling but responds to client requests allowing analysis of the failure to occur using internal diagnostics available via the mdiag command.

Related Topics

- 12.7 Problems with Individual Jobs

12.2 Logging

Moab Workload Manager provides the ability to produce detailed logging of all of its activities. This is accomplished using verbose server logging, event logging, and system logging facilities.

In this section:

12.2.1 Log Facility Configuration

The LOGFILE and/or LOGDIR parameters within the `moab.cfg` file specify the destination of this logging information. Logging information will be written in the file `<MOABHOMEDIR>/<LOGDIR><LOGFILE>` unless `<LOGDIR>` or `<LOGFILE>` is specified using an absolute path. If the log file is not specified or points to an invalid file, all logging information is directed to STDERR. However, because of the sheer volume of information that can be logged, it is not recommended that this be done while in production. By default, `LOGDIR` and `LOGFILE` are set to `log` and `moab.log` respectively, resulting in scheduler logs being written to `<MOABHOMEDIR>/log/moab.log`.

The parameter LOGFILEMAXSIZE determines how large the log file is allowed to become before it is rolled and is set to 10 MB by default. When the log file reaches this specified size, the log file is rolled. The parameter LOGFILEROLLDEPTH controls the number of old logs maintained and defaults to 3. Rolled log files have a numeric suffix appended indicating their order.

The LOGLEVEL parameter controls the verbosity of the messages recorded in logs. `LOGLEVEL` can be set to a value between 0 and 9, with 0 being the least verbose and 9 being the most verbose.

If a problem is detected, you might want to increase the `LOGLEVEL` value to get more details. However, doing so will cause the logs to roll faster and will also cause a lot of possibly unrelated information to clutter up the logs. Also be aware of the fact that high `LOGLEVEL` values results in large volumes of possibly unnecessary file I/O to occur on the scheduling machine. Consequently, it is not recommended that high `LOGLEVEL` values be used unless tracking a problem or similar circumstances warrant the I/O cost.

If high log levels are desired for an extended period of time and your Moab home directory is located on a network file system, performance may be improved by moving your log directory to a local file system using the `LOGDIR` parameter.

A final log related parameter is `LOGFACILITY`. This parameter can be used to focus logging on a subset of scheduler activities. This parameter is specified as a list of one or more scheduling facilities as listed in Appendix A: Moab Parameters.

Example 12-1:

```
# moab.cfg
# allow up to 30 100MB logfiles
LOGLEVEL          3
LOGDIR            /var/tmp/moab
LOGFILEMAXSIZE    100000000
LOGFILEROLLDEPTH  30
```

12.2.2 Standard Log Format

Each log event line follows a standard, tab-delimited log format: `timestamp <tab> thread ID <tab> visibility <tab> origin <tab> event code <tab> scope IDs <tab> message`

Field	Description
`timestamp`	Timestamps are given in local time, in ISO 8601 format, with a 4-digit time zone offset suffix. For example, `2025-01-27T15:18:30.000-0700`.
`thread ID`	The ID of the thread that is producing the log output.
`visibility`	Visibility is either a severity (FATAL, ERROR, WARNING, INFO) or a trace level (TRACE1, TRACE2, TRACE3).
`origin`	Origin is where the log event came from.
`event code`	The event code provides a way to determine what kind of event happened. For a full list of event codes, see Appendix C: Moab Event Dictionary. When there is no matching event, this field will instead show the hex value of the log level required to display that log statement.
`scope IDs`	The scope ID associates the event with a specific job or service.
`message`	Messages can give details about the event and possibly some action information to resolve issues.

Example 12-2:

```
2024-08-15T05:26:18.108-0600   846     TRACE1   MQueue.c:MQueueCheckStatus:3081     0
     MQueueCheckStatus()
2024-08-15T05:26:18.108-0600   846     TRACE1   MNode.c:MNodeCheckStatus:949        0
MNodeCheckStatus()
```

```
2024-08-15T05:26:18.108-0600   846     TRACE1  MVC.c:MVCHarvestVCs:2911            0
Checking for VCs to harvest
2024-08-15T05:26:18.108-0600   846     TRACE1  MSU.c:MUClearChild:5301             0
MUClearChild(PID)
2024-08-15T05:26:18.108-0600   846     INFO    MSysMainLoop.c:MSysMainLoop:1059
0x1002a14      Scheduling complete. Sleeping for 60 seconds.
2024-08-15T05:26:18.108-0600   846     TRACE1  MSchedStats.c:MSchedUpdateStats:36  0
MSchedUpdateStats()
2024-08-15T05:26:18.108-0600   846     INFO    MSchedStats.c:MSchedUpdateStats:45
0x100a9da      Iteration: 23; scheduling time: 0.00 seconds.
2024-08-15T05:26:18.108-0600   846     TRACE1  MRsv.c:MRsvUpdateStats:605          0
MRsvUpdateStats()
2024-08-15T05:26:18.108-0600   846     TRACE1  MSchedStats.c:MSchedUpdateStats:164 0
current util[23]:  0/1d (0.002f%)  PH: 0.072f%  active jobs: 0 of 0 (completed: 6217)
2024-08-15T05:26:18.109-0600   846     INFO    MSysMainLoop.c:MSysMainLoop:1138
0x1000193      scheduler:Moab  A scheduler iteration is ending.
```

12.2.3 Searching Moab Logs

While major failures are reported via the mdiag -S command, these failures can also be uncovered by searching the logs using the *grep* command as in the following:

```
> grep -E "WARNING|ALERT|ERROR" moab.log
```

On a production system working normally, this list usually includes some ALERT and WARNING messages. The messages are usually self-explanatory, but if not, viewing the log can give context to the message.

If a problem is occurring early when starting the Moab scheduler (before the configuration file is read) Moab can be started up using the `-L <LOGLEVEL>`flag. If this is the first flag on the command line, then the `LOGLEVEL` is set to the specified level immediately before any setup processing is done and additional logging is recorded.

If problems are detected in the use of one of the client commands, the client command can be re-issued with the `--loglevel=<LOGLEVEL>` command line argument specified. This argument causes log information to be written to STDERR as the client command is running. As with the server, <LOGLEVEL> values from 0 to 9 are supported.

The `LOGLEVEL` can be changed dynamically by use of the mschedctl -m command, or by modifying the `moab.cfg` file and restarting the scheduler. Also, if the scheduler appears to be hung or is not properly responding, the log level can be incremented by one by sending a `SIGUSR1` signal to the scheduler process. Repeated `SIGUSR1`signals continue to increase the log level. The `SIGUSR2` signal can be used to decrease the log level by one.

If an unexpected problem does occur, save the log file as it is often very helpful in isolating and correcting the problem.

12.2.4 Event Logs

Major events are reported to both the Moab log file and the Moab event log. By default, the event log is maintained in the statistics directory and rolls on a daily basis, using the naming convention `events.WWW_MMM_DD_YYYY` as in `events.Tue_Oct_18_2024`.

In this topic:

12.2.4.A Event Log Format

The event log contains information about major job, reservation, node, and scheduler events and failures and reports this information in the following format:

```
<EVENTTIME> <EPOCHTIME>:<EID> <OBJECT> <OBJECTID> <EVENT> <DETAILS>
```

Example 12-3:

```
VERSION 500
07:03:21 110244322:0 sched clusterA   start
07:03:26 110244327:1 rsv   system.1   start   1124142432 1324142432 2 2 0.0 2342155.3
node1|node2 NA RSV=%=system.1=
07:03:54 110244355:2 job   1413       end     8 16 llw mcc 432000 Completed [batch:1]
11 08708752 1108703981 ...
07:04:59 110244410:3 rm    base       failure cannot connect to RM
07:05:20 110244431:4 sched clusterA   stop    admin
...
```

The parameter RECORDEVENTLIST can be used to control which events are reported to the event log. See the sections on job and reservation trace format for more information regarding the values reported in the details section for those records.

Record Type Specific Details Format

The format for each record type is unique and is described in the following table:

Record Type	Event Types	Description
gevent	See 10.9 Enabling Generic Events for gevent information.	Generic events are included within node records. See node detail format that follows.
job	`JOBCANCEL`,	See 14.3 Workload Event Format.

Record Type	Event Types	Description

	`JOBCHECKPOINT`, `JOBEND`, `JOBHOLD`, `JOBMIGRATE`, `JOBMODIFY`, `JOBPREEMPT`, `JOBREJECT`, `JOBRESUME`, `JOBSTART`, `JOBSUBMIT`
	`AM*`
		See 5.5.8 Accounting Events for information about the `AM*` event types.

node	`NODEDOWN`, `NODEFAILURE`, `NODEUP`	The following fields are displayed in the event file in a space-delimited line as long as Moab has information pertaining to it: state, partition, disk, memory, maxprocs, swap, os, rm, nodeaccesspolicy, class, and message, where state is the node's current state and message is a human readable message indicating reason for node state change.
rm	`RMDOWN`, `RMPOLLEND`, `RMPOLLSTART`, `RMUP`	Human readable message indicating reason for resource manager state change. For `SCHEDCOMMAND`, only create/modify commands are recorded. No record is created for general list/query commands. `ALLSCHEDCOMMAND` does the same thing as `SCHEDCOMMAND`, but it also logs info query commands.
trigger	`TRIGEND`, `TRIGFAILURE`, `TRIGSTART`	`<ATTR>="<VALUE>"[ <ATTR>="<VALUE>"]...` Where `<ATTR>` is one of the following: actiondata, actiontype, description, ebuf, eventtime, eventtype, flags, name, objectid, objecttype, obuf, offset, period, requires, sets, threshold, timeout, and so forth. See 17.1 About Object Triggers for more information.

12.2.4.B Exporting Events in Real-Time

Moab event information can be exported to external systems in real-time using the ACCOUNTINGINTERFACEURL parameter. When set, Moab activates this URL each time one of the default events or one of the events specified by the RECORDEVENTLIST parameter occurs.

While various protocols can be used, the most common protocol is `exec`, which indicates that Moab should launch the specified tool or script and pass in event information as command line arguments. This tool can then select those events and fields of interest and re-direct them as appropriate providing significant flexibility and control to the organization.

Exec Protocol Format

When a URL with an `exec` protocol is specified, the target is launched with the event fields passed in as STDIN. These fields appear exactly as they do in the event logs with the same values and order.

The `tools/sql` directory included with the Moab distribution contains `event.create.sql.pl`, a sample accounting interface processing script that can be used as a template.

12.2.5 Enabling Syslog

In addition to the log file, the Moab scheduler can report events it determines to be critical to the UNIX syslog facility via the daemon facility using priorities ranging from `INFO` to `ERROR`. See the parameter USESYSLOG. The verbosity of this logging is not affected by the LOGLEVEL parameter. In addition to errors and critical events, user commands that affect the state of the jobs, nodes, or the scheduler may also be logged to syslog. Moab syslog messages are reported using the INFO, NOTICE, and ERR syslog priorities.

By default, messages are logged to syslog's user facility. However, using the `USESYSLOG` parameter, Moab can be configured to use any of the following:

- `user`
- `daemon`
- `local0`
- `local1`
- `local2`
- `local3`
- `local4`
- `local5`
- `local6`
- `local7`

12.2.6 Managing Verbosity

In very large systems, a highly verbose log may roll too quickly to be of use in tracking specific targeted behaviors. In these cases, one or more of the following approaches may be of use:

- Use the LOGFACILITY parameter to log only functions and services of interest.
- Use syslog to maintain a permanent record of critical events and failures.
- Specify higher object loglevels on jobs, nodes, and reservations of interest (such as `NODECFG[orion13] LOGLEVEL=6`).
- Increase the range of events reported to the event log using the RECORDEVENTLIST parameter.
- Review object messages for required details.
- Run Moab in monitor mode using the parameter IGNOREUSERS, IGNOREJOBS, IGNORECLASSES, or IGNORENODES.

Related Topics

- RECORDEVENTLIST parameter
- USESYSLOG parameter
- 12.4 Notifying Administrators of Failures
- 14.3 Workload Event Format
- 3.7.25 mschedctl (-L - LOG command)

12.3 Object Messages

In this section:

12.3.1 Object Message Overview

Messages can be associated with the scheduler, jobs, and nodes. Their primary use is a line of communication between resource managers, the scheduler, and end-users. When a node goes offline, or when a job fails to run, both the resource manager and the scheduler will post messages to the object's message buffer, giving the admins and end-users a reason for the failure. They can also be used as a way for different admins and users to send messages associated with the various objects. For example, an admin can set the message "Node going down for maintenance Apr/6/24 12pm," on node `node01`, which would then be visible to other admins.

12.3.2 Viewing Messages

To view messages associated with a job (either from users, the resource manager, or Moab), run the checkjob command.

To view messages associated with a node (either from users, the resource manager, or Moab), run the checknode command.

To view system messages, use the mschedctl -l message command.

To view the messages associated with a credential, run the mcredctl -c command.

12.3.3 Creating Messages

To create a message use the mschedctl -c message `<STRING> [-o <OBJECTTYPE>:<OBJECTID>] [-w <ATTRIBUTE>=<VALUE>[-w ...]]` command.

The `<OBJECTTYPE>` can be one of the following:

- node
- job
- rsv
- user
- acct
- qos
- class
- group

The `<ATTRIBUTE>` can be one of the following:

- owner
- priority
- expiretime
- type

Valid types include:

- annotation
- other
- hold
- pendactionerror

12.4 Notifying Administrators of Failures

In this section:

12.4.1 Enabling Administrator Email

In the case of certain events, Moab can automatically send email to admins. To enable mail notification, the MAILPROGRAM parameter must be set to `DEFAULT` or point to the locally available mail client. With this set, policies such as JOBREJECTPOLICY will send email to admins if set to a value of `MAIL`.

12.4.2 Handling Events with the Notification Routine

Moab possesses a primitive event management system through the use of the notify program. The program is called each time an event of interest occurs. Currently, most events are associated with failures of some sort but use of this facility need not be limited in this way. The NOTIFICATIONPROGRAM parameter enables you to specify the name of the program to run. This program is most often locally developed and designed to take action based on the event that has occurred. The location of the notification program can be specified as a relative or absolute path. If a relative path is specified, Moab looks for the notification relative to the `$(INSTDIR)/tools` directory. In all cases, Moab verifies the existence of the notification program at start up and disables it if it cannot be found or is not executable.

The notification program's action can include steps such as reporting the event via email, adjusting scheduling parameters, rebooting a node, or even recycling the scheduler.

For most events, the notification program is called with command line arguments in a simple `<EVENTTYPE>:<MESSAGE>` format. The following event types are currently enabled:

Event Type	Format	Description
JOBCORRUPTION	`<MESSAGE>`	An active job is in an unexpected state or has one or more allocated nodes that are in unexpected states.
JOBHOLD	`<MESSAGE>`	A job hold has been placed on a job.
JOBWCVIOLATION	`<MESSAGE>`	A job has exceeded its wallclock limit.
RESERVATIONCORRUPTION	`<MESSAGE>`	Reservation corruption has been detected.
RESERVATIONCREATED	`<RSVNAME> <RSVTYPE> <NAME> <PRESENTTIME> <STARTTIME> <ENDTIME> <NODECOUNT>`	A new reservation has been created.
RESERVATIONDESTROYED	`<RSVNAME> <RSVTYPE> <PRESENTTIME> <STARTTIME> <ENDTIME> <NODECOUNT>`	A reservation has been destroyed.
RMFAILURE	`<MESSAGE>`	The interface to the resource manager has failed.

Perhaps the most valuable use of the notify program stems from the fact that additional notifications can be easily inserted into Moab to handle site specific issues. To do this, locate the proper block routine, specify the correct conditional statement, and add a call to the routine `notify(<MESSAGE>);`.

Related Topics

- JOBREJECTPOLICY parameter
- MAILPROGRAM parameter
- 12.2.4 Event Logs

12.5 Issues with Client Commands

In this section:

12.5.1 Client Overview

Moab client commands are implemented as links to the executable `mclient`. When a Moab client command runs, the client executable determines the name under which it runs and behaves accordingly. At the time Moab was configured, a home directory was specified. The Moab client attempts to open the configuration file `moab.cfg`, in the `etc/` folder of this home directory on the node where the client command executes. This means that the home directory specified at configure time must be available on all hosts where the Moab client commands are executed. This also means that a `moab.cfg` file must be available in the `etc/` folder of this home directory. When the clients open this file, they will try to load the `SCHEDCFG` parameter to determine how to contact the Moab server.

The home directory value specified at configure time can be overridden by creating an `/etc/moab.cfg` file or by setting the MOABHOMEDIR environment variable.

Once the client has determined where the Moab server is located, it creates a message, adds an encrypted checksum, and sends the message to the server. The Moab client and Moab server must use a shared secret key for this to work. When the Moab server receives the client request and verifies the message, it processes the command and returns a reply.

12.5.2 Diagnosing Client Problems

The easiest way to determine where client failures are occurring is to use built-in Moab logging. On the client side, use the `--loglevel` flag. For example:

```
> showq --loglevel=9
```

This will display verbose logging information regarding the loading of the configuration file, connecting to the Moab server, sending the request, and receiving a response.

This information almost always reveals the source of the problem. If it does not, the next step is to look at the Moab server side logs; this is done using the following steps:

1. Stop Moab scheduling so that the only activity is handling Moab client requests:

   ```
   > mschedctl -s
   ```

2. Set the logging level to *very* verbose:

   ```
   > mschedctl -m loglevel 7
   ```

3. Watch Moab activity:

   ```
   > tail -f log/moab.log | more
   ```

4. Now, in a second window, issue any failing client command, such as showq.

The `moab.log` file will record the client request and any reasons it was rejected.

12.6 Tracking System Failures

In this section:

12.6.1 System Failures

The scheduler has a number of dependencies that may cause failures if not satisfied. These dependencies are in the areas of disk space, network access, memory, and processor utilization.

In this topic:

12.6.1.A Disk Space

The scheduler uses a number of files. If the file system is full or otherwise inaccessible, the following behaviors might be noted:

Unavailable File	Behavior
moab.pid	Scheduler cannot perform single instance check.
moab.ck*	Scheduler cannot store persistent record of reservations, jobs, policies, summary statistics, and so forth.
moab.cfg /moab.dat	Scheduler cannot load local configuration.
log/*	Scheduler cannot log activities.
stats/*	Scheduler cannot write job records.

When possible, configure Moab to use local disk space for configuration files, statistics files, and logs files. If any of these files are located in a networked file system (such as NFS, DFS, or AFS) and the network or file server experience heavy loads or failures, Moab server may appear sluggish or unresponsive and client command might fail. Use of local disk space eliminates susceptibility to this potential issue.

12.6.1.B Network

The scheduler uses a number of socket connections to perform basic functions. Network failures may affect the following facilities:

Network Connection	Behavior
scheduler client	Scheduler client commands fail.
resource manager	Scheduler is unable to load/update information regarding nodes and jobs.

Network Connection	Behavior
accounting manager	Scheduler is unable to validate account access or reserve/debit account balances.

12.6.1.C Memory

Depending on cluster size and configuration, the scheduler may require up to 120 MB of memory on the server host. If inadequate memory is available, multiple aspects of scheduling might be negatively affected. The scheduler log files should indicate if memory failures are detected and mark any such messages with the ERROR or ALERT keywords.

12.6.1.D Processor Utilization

On a heavily loaded system, the scheduler may appear sluggish and unresponsive. However, no direct failures should result from this slowdown. Indirect failures may include timeouts of peer services (such as the resource manager or accounting manager) or timeouts of client commands. All timeouts should be recorded in the scheduler log files.

12.6.2 Internal Errors

The Moab scheduling system contains features to assist in diagnosing internal failures. If the scheduler exits unexpectedly, the scheduler logs may provide information regarding the cause. If no reason can be determined, use of a debugger might be required.

Logs

The first step in any exit failure is to check the last few lines of the scheduler log. In many cases, the scheduler may have exited due to misconfiguration or detected system failures. The last few lines of the log should indicate why the scheduler exited and what changes are required to correct the situation. If the scheduler did not intentionally exit, increasing the LOGLEVEL parameter to `7`, or higher, may help isolate the problem.

12.6.3 Reporting Failures

If an internal failure is detected on your system, the information of greatest value to developers in isolating the problem will be the output of the gdb where subcommand and a printout of all variables associated with the failure. In addition, a level 7 log covering the failure can also help in determining the environment that caused the failure. If you encounter such and require assistance, submit a ticket at Support Portal.

If you do not already have a support username and password, create a free account to request a support ticket

12.7 Problems with Individual Jobs

To determine why a particular job will not start, there are several helpful commands:

checkjob -v

`checkjob` evaluates the ability of a job to start immediately. Tests include resource access, node state, job constraints (such as startdate, taskspernode, and QoS). Additionally, command line flags can be specified to provide more information.

Flag	Description
-l `<POLICYLEVEL>`	Evaluates impact of throttling policies on job feasibility.
-n `<NODENAME>`	Evaluates resource access on specific node.
-r `<RESERVATION_LIST>`	Evaluates access to specified reservations.

checknode

Displays detailed status of node.

mdiag -b

Displays various reasons job is considered blocked or non-queued.

mdiag -j

Displays high level summary of job attributes and performs sanity check on job attributes/state.

showbf -v

Determines general resource availability subject to specified constraints.

12.8 Diagnostic Scripts

Moab Workload Manager provides diagnostic scripts that can help aid in monitoring the state of the scheduler, resource managers, and other important components of the cluster software stack. These scripts can also be used to help diagnose issues that might need to be resolved with the help of Adaptive Computing support staff. This section introduces available diagnostic scripts.

In this section:

12.8.1 support-diag.py

The `support-diag.py` script has a two-fold purpose. First, it can be used by a Moab trigger or cron job to create a regular snapshot of the state of Moab. The script captures the output of several Moab diagnostic commands (such as *`showq`*, *`mdiag -n`*, and *`mdiag -S`*), gathers configuration/log files, and records pertinent operating system information. This data is then compressed in a time-stamped tarball for easy long-term storage.

Second, the script provides Adaptive Computing support personnel with a complete package of information that can be used to help diagnose configuration issues or system bugs. After capturing the state of Moab, the resulting tarball can be sent to your Adaptive Computing support contact for further diagnosis.

The script asks you for the trouble ticket number, `-t <TICKET#>`, or `-n`. If you chose to enter `-t <TICKET#>` the script uploads your support diagnostic output to Adaptive Computing Customer Support. The upload and ticket number request can be prevented using the `-n` option.

Synopsis

`support-diag.py` [`<options>`]

Arguments

Argument	Description
`-h, --help`	Show this help message and exit.
`-q, --diag-torque-off, --without-torque`	Disable Torque diagnostics.

Argument	Description
`-p TMPDIR, --tmp-dir=TMPDIR`	Use a different tmp directory to store output.
`-n, --no-upload`	Do not upload to Adaptive Computing.
`-t TICKET#`	Support ticket number.
`-f, --full-mode`	Gather additional logs, stats and, `moab.db` files.
`-u TIMEOUT, --moab-timeout=TIMEOUT`	Define Moab command timeout (default: 300 seconds).
`-d, --debug-mode`	`support-diag` print debug variables.
`-o, --offline-mode`	Gather offline logging only.
`-r, --ftp`	Use ftp instead of scp.
`-V, --version`	Print version information.

12.8.2 support.diag.pl

This script is deprecated. Use the `support-diag.py` script instead.

The `support.diag.pl` script has a two-fold purpose. First, it can be used by a Moab trigger or cron job to create a regular snapshot of the state of Moab. The script captures the output of several Moab diagnostic commands (such as *`showq`*, *`mdiag -n`*, and *`mdiag -S`*), gathers configuration/log files, and records pertinent operating system information. This data is then compressed in a time-stamped tarball for easy long-term storage.

The second purpose of the `support.diag.pl` script is to provide Adaptive Computing support personnel with a complete package of information that can be used to help diagnose configuration issues or system bugs. After capturing the state of Moab, the resulting tarball could be sent to your Adaptive Computing support contact for further diagnosis.

The `support.diag.pl` will ask you for the trouble ticket number then guide you through the process of uploading the data to Adaptive Computing Customer Support. The uploading and ticket number request can be prevented using the `--no-upload` and `--`

`support-ticket=<SUPPORT_TICKET_ID>` flags detailed in the Arguments table that follows.

Synopsis

`support.diag.pl` [`--include-log-lines=<NUM>`] [`--diag-torque`]

Arguments

Argument	Description
`--include-log-lines=<NUM>`	Instead of including the entire `moab.log` file, only the last `<NUM>` lines are captured in the diagnostics.
`--diag-torque`	Diagnostic commands pertinent to the Torque resource manager are included.
`--no-upload`	Prevents the system from asking the user if they want to upload the tarball to Adaptive Computing Customer Support.
`--support-ticket=<SUPPORT_TICKET_ID>`	Prevents the system from asking the user for a support ticket number.

Chapter 13: Improving User Effectiveness

In this chapter:

13.1 User Feedback Loops

Almost invariably, real world systems outperform simulated systems, even when all policies, reservations, workload, and resource distributions are fully captured and emulated. What is it about real world usage that is not emulated via a simulation? The answer is the user feedback loop, the impact of users making decisions to optimize their level of service based on real time information.

A user feedback loop is created any time information is provided to a user that modifies job submission or job management behavior. As in a market economy, the cumulative effect of many users taking steps to improve their individual scheduling performance results in better job packing, lower queue time, and better overall system utilization. Because this behavior is beneficial to the system at large, system admins and management should encourage this behavior and provide the best possible information to them.

There are two primary types of information that help users make improved decisions: cluster wide resource availability information and per-job resource utilization information.

In this section:

13.1.1 Improving Job Size/Duration Requests

Moab provides a number of informational commands that help users make improved job management decisions based on real-time cluster wide resource availability information.

These commands include showbf, showstats -f, and showq. Using these commands, a user can determine what resources are available and what job configurations statistically receive the best scheduling performance.

13.1.2 Improving Resource Requirement Specification

A job's resource requirement specification tells the scheduler what type of compute nodes are required to run the job. These requirements may state that a certain amount of memory is required per node or that a node has a minimum processor speed. At many sites, users will determine the resource requirements needed to run an initial job. Then, for the next several years, they will use the same basic batch command file to run all of their remaining jobs even though the resource requirements of their subsequent jobs may be very different from their initial run. Users often do not update their batch command files even though these constraints may be unnecessarily limiting the resources available to their jobs for two reasons: (1) users do not know how much their performance will improve if better information were provided and (2) users do not know exactly what resources their jobs are using and are afraid to lower their job's resource requirements since doing so might cause their job to fail.

To help with determining accurate per job resource utilization information, Moab provides the FEEDBACKPROGRAM facility. This tool enables sites to send detailed resource utilization information back to users via email, to store it in a centralized database for report preparation, or use it in other ways to help users refine their batch jobs.

13.2 User Level Statistics

Besides displaying job queues, end-users can display a number of their own statistics. The showstats -u `<USER_ID>` command displays current and historical statistics for a user as seen below:

```
$ showstats -u john
statistics initialized Wed Dec 31 17:00:00

         |------ Active ------|--------------------------------- Completed -----------
------------------------|
user       Jobs Procs ProcHours Jobs    %     PHReq    %     PHDed    %    FSTgt   AvgXF
MaxXF   AvgQH  Effic  WCAcc
john          1     1     30.96    9   0.00  300.0   0.00  148.9   0.00 -----    0.62
0.00   4.33 100.00  48.87
```

Users can query available system resources with the showbf command. This can aid users in requesting node configurations that are idle. Also, users can use the checkjob command to determine what parameter(s) are restricting their job from running. Moab performs better with more accurate wallclock estimates.

Moab must use an ODBC-compliant database to report statistics with Viewpoint reports.

13.3 Enhancing Wallclock Limit Estimates

As explained in the previous section, showstats -u <USER_ID> reports statistics for a given user. The *showstats -u* command can be accessed by all users. They can use fields such as PHReq, PHDed, or WCAcc to gauge wallclock estimates. Accurate wallclock estimates allow a job to be scheduled as soon as possible in a slot that it will fit in. Low or high estimates can cause a job to be scheduled in a less favorable position.

13.4 Job Start Time Estimates

In this section:

13.4.1 Example

Each user can use the showstart command to display estimated start and completion times. The following example illustrates a typical response from issuing this command:

```
> showstart orion.13762
job orion.13762 requires 2 procs for 0:33:20
Estimated Rsv based start in                 1:04:55 on Fri Jul 15 12:53:40
Estimated Rsv based completion in            2:44:55 on Fri Jul 15 14:33:40
Estimated Priority based start in            5:14:55 on Fri Jul 15 17:03:40
Estimated Priority based completion in       6:54:55 on Fri Jul 15 18:43:40
Estimated Historical based start in         00:00:00 on Fri Jul 15 11:48:45
Estimated Historical based completion in     1:40:00 on Fri Jul 15 13:28:45
Best Partition: fast
```

13.4.2 Estimation Types

In this topic:

13.4.2.A Reservation-Based Estimates

Reservation-based start time estimation incorporates information regarding current administrative, user, and job reservations to determine the earliest time the specified job can allocate the needed resources and start running. In essence, this estimate indicates the earliest time the job will start, assuming this job is the highest priority job in the queue.

For reservation-based estimates, the information provided by this command is more highly accurate if the job is highest priority, if the job has a reservation, or if the majority of the jobs that are of higher priority have reservations. Consequently, site admins wanting to make decisions based on this information might want to consider using the RESERVATIONDEPTH parameter to increase the number of priority-based reservations. This can be set so that most, or even all, idle jobs receive priority reservations and make the results of this command generally useful. The only caution of this approach is that increasing the `RESERVATIONDEPTH` parameter more tightly constrains the decisions of the scheduler and may result in slightly lower system utilization (typically less than 8% reduction).

13.4.2.B Backlog/Priority Estimates

Priority-based job start analysis determines when the queried job will fit in the queue and determines the estimated amount of time required to complete the jobs currently running or scheduled to run before this job can start.

In all cases, if the job is running, this command returns the time the job starts. If the job already has a reservation, this command returns the start time of the reservation.

13.4.2.C Historical Estimates

Historical analysis uses historical queue times for jobs that match a similar processor count and job duration profile. This information is updated on a sliding window that is configurable within `moab.cfg`.

Related Topics

- 3.7.34 showstart (command)

13.5 Providing Resource Availability Information

Moab provides commands to allow the user to query available resources. The showbf command displays what resources are available for immediate use. Using different command line parameters, such as `-m`, `-n`, and `-q` enables the user to query resources based on memory, nodecount, or QoS respectively.

13.6 Collecting Performance Information on Individual Jobs

Individual job information can be collected from the statistics file in STATDIR, which contains start time, end time, end state, QoS requested, QoS delivered, and so forth for different jobs. Also, Moab optionally provides similar information to a site's feedback program. See 18.1 User Feedback Facility for more information about the feedback program.

Chapter 14: Cluster Analysis and Testing

In this chapter:

Moab has a number of unique features that allow site admins to visualize current cluster behavior and performance, safely evaluate changes on production systems, and analyze probable future behaviors within a variety of environments.

These capabilities are enabled through a number of Moab facilities that might not appear to be closely related at first. However, taken together, these facilities allow organizations the ability to analyze their cluster without the losses associated with policy conflicts, unnecessary downtime, and faulty systems middleware.

14.1 Testing New Releases and Policies

In this section:

14.1.1 Moab Evaluation Modes

In this topic:

14.1.1.A MONITOR Mode

Moab supports a scheduling mode called `MONITOR`. In this mode, the scheduler initializes, contacts the resource manager and other peer services, and conducts scheduling cycles exactly as it would if running in `NORMAL` or production mode. Jobs are prioritized, reservations created, policies and limits enforced, and admin and end-user commands enabled. The key difference is that although live resource management information is loaded, `MONITOR` mode disables Moab's ability to start, preempt, cancel, or otherwise modify jobs or resources. Moab continues to attempt to schedule exactly as it would in `NORMAL` mode, but its ability to actually impact the system is disabled. Using this mode, you can quickly verify correct resource manager configuration and scheduler operation. This mode can also be used to validate new policies and constraints. In fact, Moab can be run in `MONITOR` mode on a production system while another scheduler or even another version of Moab is running on the same system. This unique ability can allow new versions and configurations to be fully tested without any exposure to potential failures and with no cluster downtime.

To run Moab in `MONITOR` mode, simply set the `MODE` attribute of the `SCHEDCFG` parameter to `MONITOR` and start Moab. Normal scheduler commands can be used to evaluate configuration and performance. Diagnostic commands can be used to look for any potential issues. Further, the Moab log file can be used to determine which jobs Moab attempted to start, and which resources Moab attempted to allocate.

If another instance of Moab is running in production and a site admin wants to evaluate an alternative configuration or new version, this is easily done but care should be taken to avoid conflicts with the primary scheduler. Potential conflicts include statistics files, logs, checkpoint files, and user interface ports. One of the easiest ways to avoid these conflicts is to create a new test directory with its own log and statistics subdirectories. The new `moab.cfg` file can be created from scratch or based on the existing `moab.cfg` file already in use. In either case, make certain that the `PORT` attribute of the `SCHEDCFG` parameter differs from that used by the production scheduler by at least two ports. If testing with the production binary executable, the MOABHOMEDIR environment variable should be set to point to the new test directory to prevent Moab from loading the production `moab.cfg` file.

14.1.1.B TEST Mode

`TEST` mode behaves much like `MONITOR` mode with the exception that Moab will log the scheduling actions it would have taken to the `stats/<DAY>.events` file. Using this file, sites can determine the actions Moab would have taken if running in `NORMAL` mode and verify all actions are in agreement with expected behavior.

14.1.1.C INTERACTIVE Mode

`INTERACTIVE` mode allows for evaluation of new versions and configurations in a manner different from `MONITOR` mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and requests permission to complete it. For example, before starting a job, Moab will post something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The admin must specifically accept each command request after verifying it correctly meets desired site policies. Moab will then execute the specified command. This mode is highly useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to `NORMAL` after verifying correct behavior.

14.1.2 Testing New Releases

By default, Moab runs in a mode called `NORMAL`, which indicates that it is responsible for the cluster. It loads workload and resource information, and is responsible for managing that workload according to mission objectives and policies. It starts, cancels, preempts, and modifies jobs according to these policies.

If Moab is configured to use a mode called TEST, it loads all information, performs all analysis, but, instead of actually starting or modifying a job, it merely logs the fact that it would have done so. A test instance of Moab can run at the same time as a production instance of Moab. A test instance of Moab can also run while a production scheduler of another type (such as PBS) is simultaneously running. This multi-scheduler ability allows stability and performance tests to be conducted that can help answer the following questions:

- What impact do Moab services have on network, processor, and memory load?
- What impact do Moab services have on the underlying resource manager?
- Is Moab able to correctly import resource, workload, policy, and credential information from the underlying resource manager?
- Are Moab's logged scheduling decisions in line with mission objectives?

In test mode, all of Moab's commands and services operate normally allowing the use of client commands to perform analysis. In most cases, the mdiag command is of greatest value, displaying loaded values, and reporting detected failures, inconsistencies, and object corruption. The following table highlights the most common diagnostics performed:

Command	Object
mdiag -n	Compute nodes, storage systems, network systems, and generic resources
mdiag -j	Applications and static jobs
mdiag -u **mdiag -g** **mdiag -a**	User, group, and account credentials
mdiag -c	Queues and policies
mdiag -R	Resource manager interface and performance
mdiag -S	Scheduler/system level failures introduced by corrupt information

These commands will not only verify proper scheduling objects but will also analyze the behavior of each resource manager, recording failures, and delivered performance. If any misconfiguration, corruption, interface failure, or internal failure is detected, it can be addressed in the test mode instance of Moab with no urgency or risk to production cluster activities.

14.1.3 Testing New Policies

Verifying Correct Specification of New Policies

The first aspect of verifying a new policy is verifying correct syntax and semantics. If manually editing the `moab.cfg` file, the following command can be used for validation:

```
> mdiag -C
```

This command will validate the configuration file and report any misconfiguration.

Verifying Correct Behavior of New Policies

If concern exists over the impact of a new policy, an admin can babysit Moab by putting it into INTERACTIVE mode. In this mode, Moab will schedule according to all mission objectives and policies, but before taking any action, it will request that the admin confirm the action. See the section INTERACTIVE Mode above for more information.

In this mode, only actions approved by the admin will be carried out. Once proper behavior is verified, the Moab mode can be set to `NORMAL`.

14.1.4 Moab Side-by-Side

Moab provides an additional evaluation method that allows a production cluster or other resource to be logically partitioned along resource and workload boundaries and allows different instances of Moab to schedule different partitions. The parameters IGNORENODES, IGNORECLASSES, IGNOREJOBS, and IGNOREUSERS are used to specify how the system is to be partitioned. In the following example, a small portion of an existing cluster is partitioned for temporary grid testing so that there is no impact on the production workload:

```
SCHEDCFG[prod] MODE=NORMAL SERVER=orion.cxz.com:42020
RMCFG[Torque]  TYPE=PBS
IGNORENODES    node61,node62,node63,node64
IGNOREUSERS    gridtest1,gridtest2
...
SCHEDCFG[prod] MODE=NORMAL SERVER=orion.cxz.com:42030
RMCFG[Torque]  TYPE=PBS
IGNORENODES    !node61,node62,node63,node64
IGNOREUSERS    !gridtest1,gridtest2
...
```

Two completely independent Moab servers schedule the cluster. The first server handles all jobs and nodes except for the ones involved in the test. The second server handles only test nodes and test jobs. While both servers actively talk and interact with a single Torque resource manager, the `IGNORE*` parameters cause them to not schedule, nor even see the other partition and its associated workload.

When enabling Moab side-by-side, each Moab server should have an independent home directory to prevent logging and statistics conflicts. Also, in this environment, each Moab server should communicate with its client commands using a different port as shown in the previous example.

When specifying the `IGNORENODES` parameter, the exact node names, as returned by the resource manager, should be specified.

Related Topics

- 8.3 Testing New Versions and Configurations

14.2 Testing New Middleware

Moab can be used to drive new middleware stress testing resource management systems, information services, allocation services, security services, data staging services, and other aspects. Moab is unique when compared to other stress testing tools as it can perform the tests in response to actual or recorded workload traces, performing a playback of events and driving the underlying system as if it were part of the production environment.

This feature can be used to identify scalability issues, pathological use cases, and accounting irregularities in anything from LDAP, to NIS, and NFS.

Using Moab's time management facilities, Moab can drive the underlying systems in accordance with the real recorded distribution of time, at a multiplier of real time, or as fast as possible.

In this section:

14.2.1 Analysis Aspects

The following table describes some aspects of cluster analysis that can be driven by Moab:

System	Details
Accounting Manager	Use `test` mode to drive scheduling queries, allocation debits, and reservations to accounting packages. Verify synchronization of cluster statistics and stress test interfaces and underlying databases.
On-Demand/Provisioning Services	Use Native Resource Manager mode to drive triggers and resource management interfaces to enable dynamic provisioning of hardware, operating systems, application software, and services. Test reliability and scalability of data servers, networks, and provisioning software, and the interfaces and business logic coordinating these changes.
Resource Monitoring	Use `test` or Native Resource Manager mode to actively load information from compute, network, storage, and software license managers confirming validity of data, availability during failures, and scalability.

With each evaluation, the following tests can be enabled:

- functionality
- reliability
  - hard failure
    - hardware failure - compute, network, and data failures
    - software failure - loss of software services (NIS, LDAP, NFS, database)
    - soft failure
    - network delays, full file system, dropped network packets
  - corrupt data
- performance
- determine peak responsiveness in seconds/request
- determine peak throughput in requests/second
- determine responsiveness under heavy load conditions
- determine throughput under external load conditions
  - large user base (many users, groups, accounts)
  - large workload (many jobs)
  - large cluster (many nodes)
- manageability
  - full accounting for all actions/events
  - actions/failures can be easily and fully diagnosed

If using a Native Resource Manager and you do not want to actually submit real workload, you can set the environment variable MFORCESUBMIT to allow virtual workload to be managed without ever launching a real process.

14.2.2 General Analysis

For all middleware interfaces, Moab provides built-in performance analysis and failure reporting. Diagnostics for these interfaces are available via the mdiag command.

14.2.3 Native Mode Analysis

Using Native mode analysis, organizations can run Moab in `normal` mode with all facilities fully enabled, but with the resource manager fully emulated. With a Native Resource

Manager interface, any arbitrary cluster can be emulated with a simple script or flat text file. Artificial failures can be introduced, jobs can be virtually running, and artificial performance information generated and reported.

In the simplest case, emulation can be accomplished using the following configuration:

```
SCHEDCFG[natcluster] MODE=NORMAL SERVER=test1.bbli.com
ADMINCFG[1] USERS=dev
RMCFG[natcluster] TYPE=NATIVE CLUSTERQUERYURL=file://$HOME/cluster.dat
```

The preceding configuration will load cluster resource information from the file `cluster.dat`. An example resource information file follows:

```
node01 state=idle cproc=2
node02 state=idle cproc=2
node03 state=idle cproc=2
node04 state=idle cproc=2
node05 state=idle cproc=2
node06 state=idle cproc=2
node07 state=idle cproc=2
node08 state=idle cproc=2
```

In actual usage, any number of node attributes can be specified to customize these nodes, but in this example, only the node state and node configured processors attributes are specified.

The `RMCFG` flag `NORMSTART` indicates that Moab should not actually issue a job start command to an external entity to start the job, but rather start the job logically internally only.

If it is desirable to take an arbitrary action at the start of a job, end of a job, or anywhere in between, the `JOBCFG` parameter can be used to create one or more arbitrary triggers to initiate internal or external events. The triggers can do anything from executing a script, to updating a database, to using a Web service.

Using Native Resource Manager mode, jobs can be introduced using the msub command according to any arbitrary schedule. Moab will load them, schedule them, and start them according to all site mission objectives and policies and drive all interfaced services as if running in a full production environment.

14.3 Workload Event Format

Moab workload accounting records fully describe all scheduling relevant aspects of batch jobs including resources requested and used, time of all major scheduling events (such as submission time and start time), the job credentials used, and the job execution environment. Each job trace is composed of a single line consisting of either attribute=value pairs or whitespace-delimited fields as shown in the following table. The attribute=value pairs format can be found in N.2.2 Query Workload Data Format.

Moab can be configured to provide this information in flat text tabular form or in XML format conforming to the SSS 1.0 job description specification.

In this section:

14.3.1 Workload Event Record Format

All job events (`JOBSUBMIT`, `JOBSTART`, `JOBEND`, and so forth) provide job data in a standard format as described in the following table:

Workload Event Record Format Table

Field Name	Field Index	Data Format	Default Value	Details
Event Time (Human Readable)	1	`HH:MM:SS`	-	Specifies time event occurred.
Event Time (Epoch)	2	`<epochtime>:<eventID>`	-	Specifies time event occurred and the unique event ID.
Object Type	3	job	-	Specifies record object type.
Object ID	4	`<STRING>`	-	Unique object identifier.
Object Event	5	One of `jobcancel`, `jobcheckpoint`, `jobend`, `jobfailure`, `jobhold`, `jobmigrate`, `jobpreempt`, `jobreject`, `jobresume`, `jobstart` or `jobsubmit`	-	Specifies record event type.
Nodes Requested	6	`<INTEGER>`	0	Number of nodes requested (0 = no node request count specified).
Tasks Requested	7	`<INTEGER>`	1	Number of tasks requested.
User Name	8	`<STRING>`	-	Name of user submitting job.
Group Name	9	`<STRING>`	-	Primary group of user submitting job.
Wallclock Limit	10	`<INTEGER>`	1	Maximum allowed job duration (in seconds).

Field Name	Field Index	Data Format	Default Value	Details
Job Event State	11	`<STRING>`	-	Job state at time of event.
Required Class	12	`<STRING>`	[DEFAULT: 1]	Class/queue required by job specified as square bracket list of <QUEUE>[:<QUEUEINSTANCE>] requirements. (For example: [batch:1]).
Submission Time	13	`<INTEGER>`	0	Epoch time when job was submitted.
Dispatch Time	14	`<INTEGER>`	0	Epoch time when scheduler requested job begin executing.
Start Time	15	`<INTEGER>`	0	Epoch time when job began executing. This is usually identical to Dispatch Time.
Completion Time	16	`<INTEGER>`	0	Epoch time when job completed execution.
Required Node Architecture	17	`<STRING>`	-	Required node architecture if specified.
Required Node Operating System	18	`<STRING>`	-	Required node operating system if specified.
Required Node Memory Comparison	19	One of >, >=, =, <=, <	>=	Comparison for determining compliance with required node memory.

Field Name	Field Index	Data Format	Default Value	Details
Required Node Memory	20	`<INTEGER>`	0	Amount of required configured RAM (in MB) on each node.
Required Node Disk Comparison	21	One of >, >=, =, <=, <	>=	Comparison for determining compliance with required node disk.
Required Node Disk	22	`<INTEGER>`	0	Amount of required configured local disk (in MB) on each node.
Required Node Attributes/Features	23	`<STRING>`	-	Square bracket enclosed list of node features required by job if specified. (For example: [fast][ethernet])
System Queue Time	24	`<INTEGER>`	0	Epoch time when job met all fairness policies.
Tasks Allocated	25	`<INTEGER>`	`<TASKS REQUESTED>`	Number of tasks actually allocated to job. In most cases, this field is identical to field #7, Tasks Requested.
Required Tasks Per Node	26	`<INTEGER>`	-1	Number of Tasks Per Node required by job or '-1' if no requirement specified.
QOS	27	`<STRING>[:<STRING>]`	-	QoS requested/assigned using the format `<QOS_REQUESTED>[:<QOS_DELIVERED>]`. (For example:

Field Name	Field Index	Data Format	Default Value	Details
				hipriority:bottomfeeder)
JobFlags	28	`<STRING>[:<STRING>]...`	-	Square bracket delimited list of job attributes. (For example: `[BACKFILL][PREEMPTEE]`).
Account Name	29	`<STRING>`	-	Name of account associated with job if specified.
Executable	30	`<STRING>`	-	Name of job executable if specified.
Resource Manager Extension String	31	`<STRING>`	-	Resource manager specific list of job attributes if specified. See 11.3 Resource Manager Extensions for more information.
Bypass Count	32	`<INTEGER>`	-1	Number of times job was bypassed by lower priority jobs via backfill or '-1' if not specified.
ProcSeconds Utilized	33	`<DOUBLE>`	0	Number of processor seconds actually used by job.
Partition Name	34	`<STRING>`	[DEFAULT]	Name of partition where job ran.
Dedicated Processors per Task	35	`<INTEGER>`	1	Number of processors required per task.
Dedicated Memory per Task	36	`<INTEGER>`	0	Amount of RAM (in MB) required per task.

Field Name	Field Index	Data Format	Default Value	Details
Dedicated Disk per Task	37	`<INTEGER>`	0	Amount of local disk (in MB) required per task.
Dedicated Swap per Task	38	`<INTEGER>`	0	Amount of virtual memory (in MB) required per task.
Start Date	39	`<INTEGER>`	0	Epoch time indicating earliest time job can start.
End Date	40	`<INTEGER>`	0	Epoch time indicating latest time by which job must complete.
Allocated Host List	41	`<hostname>[,<hostname>]...`	-	Comma-delimited list of hosts allocated to job. (For example: node001,node004).
Resource Manager Name	42	`<STRING>`	-	Name of resource manager if specified.
Required Host List	43	`<hostname>[,<hostname>]...`	-	List of hosts required by job. (If the job's taskcount is greater than the specified number of hosts, the scheduler must use these nodes in addition to others; if the job's taskcount is less than the specified number of hosts, the scheduler must select needed hosts from this list.)
Reservation	44	`<STRING>`	-	Name of reservation required by job if specified.

Field Name	Field Index	Data Format	Default Value	Details
Application Simulator Data	45	`<STRING>[:<STRING>]`	-	Name of application simulator module and associated configuration data. (For example: `HSM:IN=infile.txt:140000;OUT=outfile.txt:500000`).
Set Description	46	`<STRING>:<STRING>[:<STRING>]`	-	Set constraints required by node in the form `<SetConstraint>:<SetType>[:<SetList>]` where SetConstraint is one of ONEOF, FIRSTOF, or ANYOF, SetType is one of PROCSPEED, FEATURE, or NETWORK, and SetList is an optional colon-delimited list of allowed set attributes. (For example: `ONEOF:PROCSPEED:350:450:500`)
Job Message	47	`<STRING>`	-	Job messages including resource manager, scheduler, and admin messages if specified.
Job Cost	48	`<DOUBLE>`	0.0	Cost of executing job incorporating resource consumption metric, resource quantity consumed, and credential, allocated resource, and delivered QoS charge rates.
History	49	`<STRING>`	-	List of job events impacting resource allocation (XML).

Field Name	Field Index	Data Format	Default Value	Details
Utilization	50	Comma-delimited list of one or more of the following: `<ATTR>=<VALUE>` pairs, where `<VALUE>` is a double and `<ATTR>` is one of the following: `network` (in MB transferred), `license` (in license-seconds), `storage` (in MB-seconds stored), or `gmetric:<TYPE>`.	-	Cumulative resources used over life of job.
Estimate Data	51	`<STRING>`	-	List of job estimate usage.
Completion Code	52	`<INTEGER>`	-	Job exit status/completion code.
Extended Memory Load Information	53	`<STRING>`	-	**Deprecated.** Extended memory usage statistics (max, mem, avg, and so forth).
Extended CPU Load Information	54	`<STRING>`	-	Extended CPU usage statistics (max, mem, avg, and so forth).
Generic Metric Averages	55	`<STRING>`	-1	Generic metric averages.
Effective Queue Duration	56	`<INTEGER>`	-1	The amount of time, in seconds, that the job was eligible for scheduling.

Field Name	Field Index	Data Format	Default Value	Details
Job Submission Arguments	57	`<STRING>`	-	The job's submit arguments and script. This field is enabled by setting the parameter STOREJOBSUBMISSION to `TRUE`.

If a field has an empty value, Moab will use a single dash `(-)` as a placeholder in the event record.

Fields that contain a description string such as Job Message use a packed string format. The packed string format replaces white space characters such as spaces and carriage returns with a hex character representation. For example a blank space is represented as `\20`. Since fields in the event record are space delimited, this preserves the correct order and spacing of fields in the record.

Sample Workload Event

```
13:21:05 110244355 job 1413 JOBEND 20 20 josh staff 86400 Removed [batch:1] 887343658
889585185 \
889585185 889585411 ethernet R6000 AIX53 >= 256 >= 0 - 889584538 20 0 0 2 0 test.cmd \
1001 6 678.08 0 1 0 0 0 0 0 - 0 - - - - - - - - - 0.0 - - - 0 - -
```

JOBSUBMISSIONPOLICY Value	Critical Time Based Fields
NORMAL	WallClock Limit Submission Time StartTime Completion Time
CONSTANTJOBDEPTH **CONSTANTPSDEPTH**	WallClock Limit StartTime Completion Time

14.3.2 Reservation Event Records

All reservation events provide reservation data in a standard format as described in the following table:

Field Name	Field Index	Data Format	Default Value	Details
Event Time (Human)	0	`[HH:MM:SS]`	-	Specifies time event occurred.
Event Time (Epoch)	1	`<epochtime>`	-	Specifies time event occurred.
Object Type	2	rsv	-	Specifies record object type.
Object ID	3	`<STRING>`	-	Unique object identifier.
Object Event	4	One of `rsvcreate`, `rsvstart`, `rsvmodify`, `rsvfail` or `rsvend`	-	Specifies record event type.
Creation Time	5	`<EPOCHTIME>`	-	Specifies epoch time of reservation start date.
Start Time	6	`<EPOCHTIME>`	-	Specifies epoch time of reservation start date.
End Time	7	`<EPOCHTIME>`	-	Specifies epoch time of reservation end date.
Tasks Allocated	8	`<INTEGER>`	-	Specifies number of tasks allocated to reservation at event time.
Nodes Allocated	9	`<INTEGER>`	-	Specifies number of nodes allocated to reservation at event time.
Total Active Proc-	10	`<INTEGER>`	-	Specifies proc-seconds reserved resources were dedicated to one or more job at event time.

Field Name	Field Index	Data Format	Default Value	Details
Seconds				
Total Proc-Seconds	11	`<INTEGER>`	-	Specifies proc-seconds resources were reserved at event time.
Hostlist	12	`<comma-delimited list of hostnames>`	-	Specifies list of hosts reserved at event time.
Owner	13	`<STRING>`	-	Specifies reservation ownership credentials.
ACL	14	`<STRING>`	-	Specifies reservation access control list.
Comment	15	`<STRING>`	-	Specifies general human readable event message.
Command Line	16	`<STRING>`	-	Displays the command line arguments used to create the reservation (only shows on the rsvcreate event).

14.3.3 Recording Job Events

Job events occur when a job undergoes a definitive change in state. Job events include submission, starting, cancellation, migration, and completion. Some site admins do not want to use an external accounting system and use these logged events to determine their clusters' accounting statistics. Moab can be configured to record these events in the appropriate event file found in the Moab `stats/` directory. To enable job event recording for both local and remotely staged jobs, use the RECORDEVENTLIST parameter. For example:

```
RECORDEVENTLIST JOBCANCEL,JOBCOMPLETE,JOBSTART,JOBSUBMIT
...
```

This configuration records an event each time both remote and/or local jobs are canceled, run to completion, started, or submitted. The Event Logs section details the format of these records.

Related Topics

- 12.2.4 Event Logs

Chapter 15: Green Computing

SearchDataCenter.com defines green computing as the environmentally responsible use of computers and related resources. Such practices include the implementation of energy-efficient CPUs, servers, and peripherals, and reduced resource consumption and proper disposal of e-waste.

The Moab HPC Suite contains power management features that give a Moab Admin the ability to implement policies that can conserve energy and save on operational costs, often without affecting an HPC system's performance with regard to job execution times.

Effective power management means managing power or energy consumption while a compute node is actively running jobs, and when a compute node is idle. Both scenarios require different tools and policies:

- Active compute node power management is mainly performed through control of the clock frequency of the processor(s) on a compute node while a job is executing. Decreasing the clock frequency can reduce energy usage.
- Idle compute node power management is mainly performed by placing a compute node into different low-power states, such as standby and suspend, or no-power states, such as hibernate and shutdown.

In this chapter:

15.1 Green Computing Methods

In this section:

15.1.1 Moab Edition Green Features

The table below identifies the Moab power management features and/or methods available in the Moab HPC Suite:

Feature or Method
CPU Clock Frequency Control: • Moab Job Submission Option • Torque Job Submission Option • Moab Job Template Option
Manual Power Management: • Moab-based `on` and `off` states • Torque-based low-power and no-power states
Automated Power Management and Green Policies: • Moab-only global-level policies and power management for `on` and `off` states • Moab/Moab Web Services-based global, partition, and node-level policies and power management for low-power and no-power states • Green Idle Node Pool Management Policies

15.1.2 Moab Power Management Methods

Moab supports two separate and mutually-exclusive methods for managing the power state of compute nodes, which affects energy consumption. The first method enables an admin to manually power on and power off compute nodes and to create a global set of green policies that automatically perform these two functions based on specific conditions involving idle compute nodes. The second method gives an admin additional power states besides `on` and `off` and offer finer control of green policies at the global, partition, and node levels. Before delving into the theory of operation of these two separate methods, an admin must understand how Moab views power management regardless of which method is used.

In this topic:

15.1.2.A Moab View of Power Management

Moab is not aware of the actual power state of nodes. From Moab's perspective, nodes are only on or off. If Moab needs a node that is off, it issues a power-on job prior to scheduling the incoming job.

In addition, in order to schedule a job to a compute node, Moab requires the compute node's workload resource manager, which in our example is Torque, to report the compute node's state is `idle`. When the compute node's binary power state indicates `on` and the resource manager indicates the compute node's state is `idle`, Moab will schedule jobs to the compute node. Any value other than `idle` for the node's state and Moab will not schedule a job to the node. If the power state is `off`, Moab issues a power-on job as a dependency to the regular job.

Moab performs compute node power management entirely through power management resource managers, or Power RMs. Each of the two power management methods mentioned above has its own Power RM implementation. The older Moab-only method uses Python-based scripts to implement a Power RM while the newer Moab+Moab Web Services (MWS)-based method uses a Java-based MWS RM Power Management plug-in that runs much simpler Python-based scripts.

These Power RMs perform all power-related management and monitoring, meaning power state control and power state query, respectively, and only report back to Moab whether a compute node is in a state where it can run jobs (on) or not (off). All actual power state-aware control and management is performed by the Power RMs.

15.1.2.B Moab Power RMs

Adaptive Computing provides two power management methods to handle different site scenarios; mainly for site-specific security policies. The older method handles sites with a security policy that does not permit web service-based services, which can be an attack vector, or sites that do not want to run an MWS service.

The newer method uses the MWS RM Power Management plug-in feature, which enables an admin to instantiate a separate resource manager Power Management plug-in instance for different partitions, or different compute nodes for situations where different compute node hardware requires the use of different power management commands run from Python scripts.

15.1.2.C Power Management Scripts

Each power management method, old or new, employs at some point a script that enables the admin to customize power management for a site, which may be required because the working reference scripts provided by Adaptive Computing (based on OpenIPMI tools) do not use the power management commands specific to the site's vendor-provided hardware.

15.1.2.D Moab System Jobs

Moab performs power management functions through a mechanism known as system jobs. A Moab system job is a special, separately scheduled job that performs some Moab system function (e.g., power management, data-staging) that Moab executes on the Moab head node and not on a compute node. This enables Moab to apply policies such as a job wallclock estimate, etc, to system-related functions, which can aid error recovery procedures, etc.

System jobs perform internal Moab-related functions on Moab's behalf, are nearly always script-based, and usually require some customization by the Moab Admin in order to perform the needed function for the HPC system site. For example, the admin might have to modify power management scripts so they use a site's hardware vendor-specific power management commands to effect power state changes in compute nodes.

To create a system job, Moab internally submits an admin-defined script, with a path typically specified as a Moab `*URL` parameter, to itself, which it flags as a system job. Moab schedules the job and because it is flagged as a system job, executes the script on the head node. Moab submits a system job whenever it needs to send a power on or off command to a Power RM. Admins can easily recognize queued and running power management system jobs in the *`showq`* command output as their job ID has the format `id.poweron` and `id.poweroff`, where `id` is the internally generated Moab job ID number and `.poweron` and `.poweroff` are suffixes appended to the job ID number that represent Moab's on and off commands sent to Power RMs.

15.1.2.E Green Policies

Moab provides green policies that automate power management for idle compute nodes, which an admin can modify and/or configure to control the power state of compute nodes not always in use. These policies allow Moab to dynamically control the power state of compute nodes between the active running state or power-on nodes that may be needed. It also enables Moab to power-off nodes that are idle and wasting energy. Which power state such compute nodes enter depends entirely on the commands the admin configures and/or modifies in a Power RM's scripts and, for the newer Moab+MWS method, on configuration information specified for each MWS RM Power Management plug-in instance.

The green policies maintain a green idle node pool, the size of which the admin configures. As jobs start and use idle nodes from the pool, Moab replenishes the pool by performing an `on` command on those compute nodes on which it previously had performed an `off` command, therefore bringing them into the idle node pool as they enter into an active running state. When jobs finish and the pool has excess idle nodes, Moab performs an `off` command on the excess nodes, which removes them from the idle pool. Therefore, Moab maintains a pool of available idle nodes for immediate use by submitted jobs and reduces energy consumption by powering off any idle nodes in excess of the pool size.

15.1.3 Theory of Operation

Moab itself operates the same regardless of the method of power management, Moab-only or Moab+MWS, chosen. This is especially true for the green policies as Moab simply uses the configured power management method to carry out the policies. In order to know how to configure the different parts and components of each power management method so they work well together, it is necessary for a site admin to understand how the power management methods work; that is, how the components work together to implement a power management method.

In this topic:

15.1.3.A Moab-Only Method

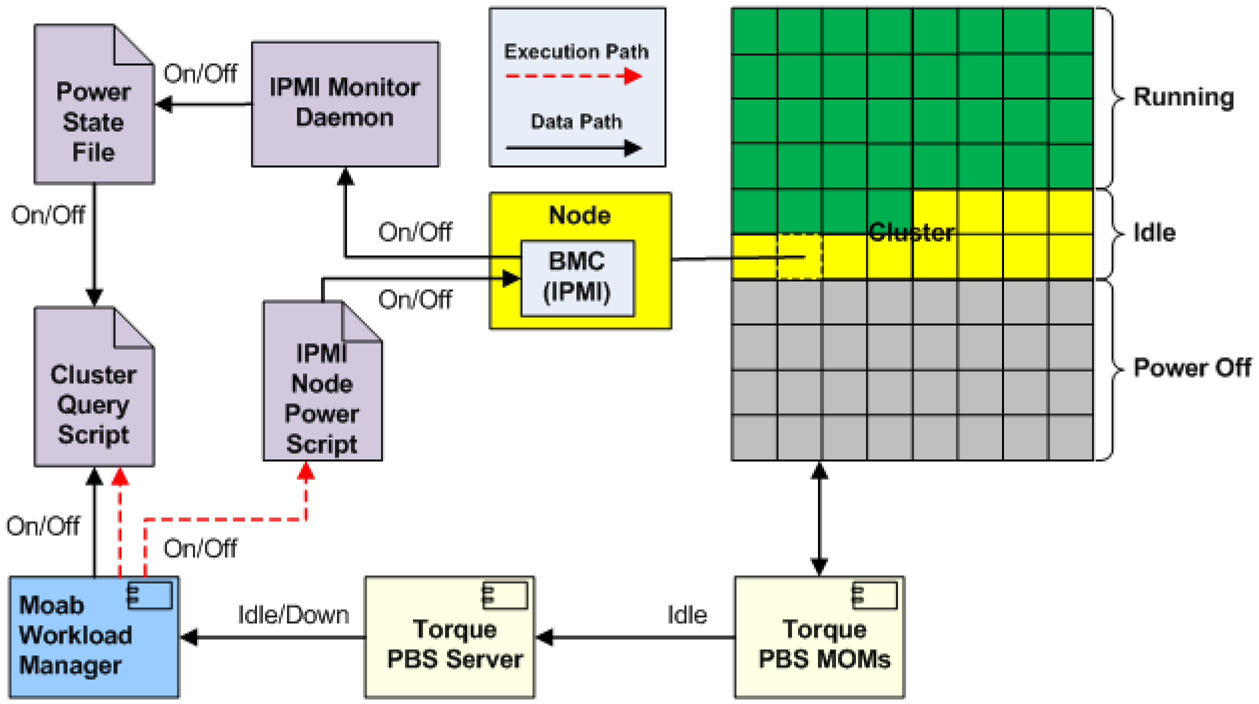
The Moab-only method has a Power RM composed entirely of Python-based scripts. The script must maintain a Power Query daemon that queries the power state of all compute nodes and saves their state for Moab to query, the actual power state query Moab runs to find out the current power state of all compute nodes, and a power state control that places compute nodes into the state of on so Moab can schedule jobs to them or into the state of off so energy consumption is minimized and operational costs reduced. The admin determines what the actual power state Moab's `off` represents by configuring the off command in the power management control script with the actual hardware vendor-supplied command that effects the desired power state (remember, Moab is not aware of actual power states).

The list below enumerates the advantages and disadvantages of the Moab-only method:

- Advantages:
  - Do not have to run the MWS service and its MongoDB database.
  - Power management command scripts execute as Moab system jobs.
  - Ability to customize the node power and cluster query power management scripts.
    - For more information on how to specify the node power control script, see the NODEPOWERURL parameter.
    - For more information on how to specify the power cluster query script, see the CLUSTERQUERYURL parameter.
  - Moab power control using `mnodectl -mpower=[on|off] <nodelist>`.
    - For more information on how to diagnose power states, see 3.7.10 mdiag -n.
- Disadvantages:
  - More complex scripts to customize.
  - Only global power management control (no partition-based or node-based).
  - Heterogeneous compute node hardware from different vendors requires more modification of the control and query scripts.
  - Reference scripts not scalable (did not take advantage of Python multi-threading).
  - Admin must maintain complex scripts that must maintain the entire cluster query information.

Moab-only Method Architecture

The following diagram shows the Moab-only Method Architecture and what occurs between its components:



The Python-based IPMI Monitor daemon script running in the background periodically polls the power state of all compute nodes through IPMI using the command customized by the admin. As it gathers power state information, it saves the information in a text file in a specific format understood by Moab (binary power state). In order to prevent race conditions, it actually writes to a temporary file and then moves the temporary file on top of the permanent file (not shown).

When Moab starts a scheduling cycle/iteration, it directly executes the Power RM's Python-based Cluster Query script that reads the permanent text file and delivers the compute node power states to Moab. Moab then performs the scheduling cycle and based on green policies and the state of the HPC cluster will run the IPMI Node Power script as a Moab system job to perform an `on` or `off` (which may be something different than a `power off`) command using the actual commands customized by the admin in the script.

15.1.3.B Moab+MWS Method

The Moab+MWS method has a Power RM composed of a MWS RM Power Management plug-in that encapsulates all power management logic, which itself uses the Torque *`pbsnodes`* command to effect compute node power state changes into low-power and no-power states of standby and suspend, and hibernate and shutdown, respectively, and the

IPMI Node Power script to effect compute node power on, power off (pull the plug) and awaken (resume active running state from low-power state). The Power RM Power Management plug-in also performs the power query daemon function identified in the Moab-only method using its built-in power management logic, therefore handling more actual power states and allowing much better power control than the Moab-only method offers.

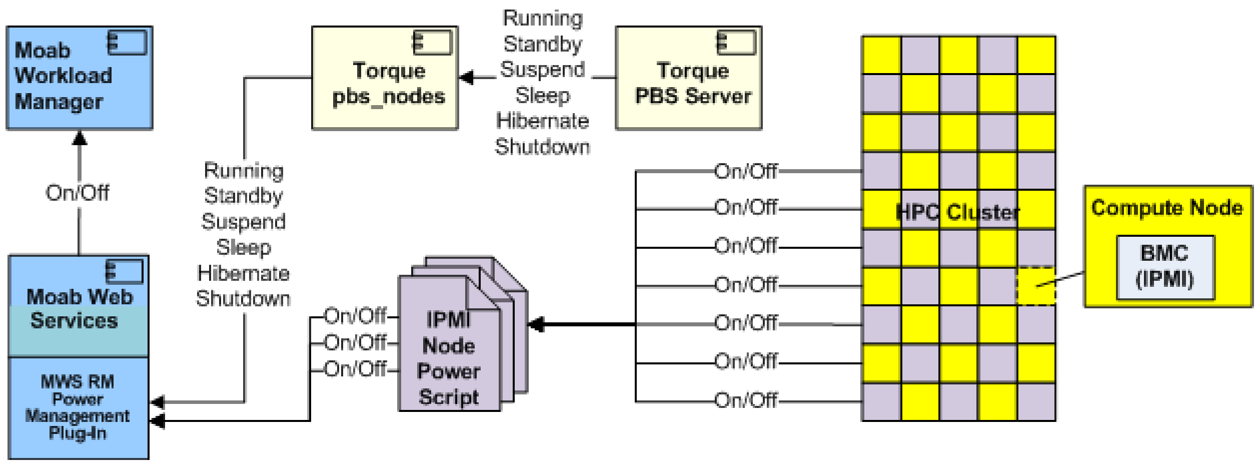
The advantages and disadvantages of the Moab+MWS-based method are enumerated below:

- Advantages:
    - More power states to choose from.
        - Low-power states of standby and suspend.
        - No-power states of hibernate and shutdown.
        - On and Off (pull the plug) power states still available.
    - Torque power control of low-power and no-power states using `pbsnodes -m <state> <nodelist>`.
        - You can view node power states with the Torque pbsnodes command.
    - Power management command scripts execute as Moab system jobs.
    - Much simpler `moab.cfg` customization and maintenance.
    - Global, partition-based, and node-based granularity for power management control.
    - Heterogeneous compute node hardware from different vendors handled by creating multiple instances of MWS RM Power Management plug-in with different configurations.
    - Reference scripts are scalable (use Python multi-threading).
    - The MWS RM architecture is easier to support DRAC, ILO, and other protocols.
- Disadvantages:
    - Must run the MWS service and its MongoDB database.
    - Configuration of the MWS RM Power Management plug-in and possible multiple instances.

The following architecture diagrams show the Moab+MWS-based method architecture and what occurs between its components.

Power State Query

The diagram below illustrates power state query:



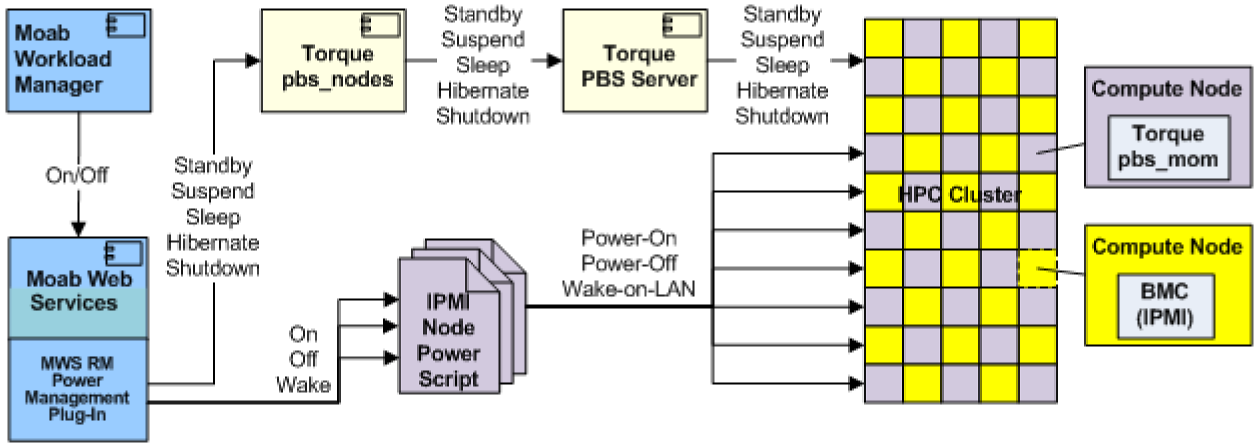
The MWS RM Power Management plug-in runs the multi-threaded Power Query script for sets of compute nodes that obtain their actual power state through IPMI, or more specifically, a hardware vendor's IPMI implementation (e.g., Dell DRAC, HP iLO, etc.), which the MWS RM Power Management plug-in saves. It also runs the Torque *pbsnodes* command to obtain the low-power or no-power states that may have been set via Torque earlier (*pbs_server* retains knowledge of any previous command to set a node's power state to one of the low-power or no-power states).

Note it is quite possible for IPMI to report `off` and Torque to report hibernate or shutdown, both of which indicate a compute node has no power, and for IPMI to report `on` and Torque to report standby or suspend, both of which indicate a compute node is in a low-power state from which it can be quickly awakened. It is also possible for IPMI to report `on` and Torque to report hibernate or shutdown, which can indicate a booting node that has not yet started the Torque *pbs_mom* daemon or a node hibernating or shutting down that has not yet powered off. The MWS RM Power Management plug-in's power management logic reconciles the IPMI and Torque reports to produce a single `on` or `off` understood by Moab, which it passes to MWS.

When Moab queries MWS for the current state information of compute nodes at the start of a scheduling cycle/iteration, MWS passes all node information including the binary power on/off Moab understands and the Torque node state, at which point Moab has the information it needs to perform green policy-based automated power management.

Moab+MWS Power State Control Interactions

The diagram below illustrates Moab+MWS power state control interactions:


Moab Workload Manager
On/Off
Moab Web Services
MWS RM Power Management Plug-In
Standby Suspend Sleep Hibernate Shutdown
Torque pbs_nodes
Standby Suspend Sleep Hibernate Shutdown
Torque PBS Server
Standby Suspend Sleep Hibernate Shutdown
On Off Wake
IPMI Node Power Script
Power-On Power-Off Wake-on-LAN
HPC Cluster
Compute Node
Torque pbs_mom
Compute Node
BMC (IPMI)

When Moab detects a condition that requires changing the power state of a compute node, usually as a result of green policies, it performs the appropriate `on` or `off` command as a system job that sends the command to MWS with a list of the host names of compute nodes that should enter an appropriate power state.

MWS interacts with the appropriate MWS RM Power Management plug-in for each compute node and passes it the `on` or `off` command. For the `off` command, the plug-in examines its configuration of what `off` means and passes the configured standby, suspend, hibernate, or shutdown command to the Torque *pbsnodes* command, or passes the configured `off` command to the Node Power script.

If the MWS RM Power Management plug-in executes the Torque *pbsnodes* command for the configured power state and requested list of compute node host names, it sends the command to the *pbs_server*, which passes the command to each compute node's *pbs_mom* daemon. The *pbs_mom* executes software to place the node into the requested state. The *pbs_server* daemon keeps the requested state in a file for each compute node, which it passes on to the MWS RM Power Management plug-in as part of a node update report.

In clusters where there is a Torque *pbs_server* and *pbs_mom* on the same machine, the admin should set the POWERPOLICY to STATIC on this node, because the *pbs_server* should not be powered down. If the *pbs_server* is powered down, Moab will be unable to get cluster query updates from all *pbs_mom*s managed by that *pbs_server*.

On all Torque nodes where *pbs_mom*s are running, the *pbs_mom* must be configured to auto-start after being rebooted. If the *pbs_mom* isn't auto-started, the *pbs_server* will not be able to determine when it has been powered up and entered an idle state, and therefore won't have the ability to inform Moab on a cluster query the node is idle. Refer to '(Optional) Startup/Shutdown Service Script for Torque/Moab' in the *Torque Resource Manager Administrator Guide* for details on how to have the *pbs_mom* auto-start on boot.

When the MWS RM Power Management plug-in executes the Node Power script for the configured `off` power state and requested list of compute node host names, the script executes its IPMI `on` command (whatever the admin configured in the script) that tells the node's baseboard management controller (BMC) to power off the node.

When the MWS RM Power Management plug-in receives the `on` command from Moab via MWS, it checks the internal power state of each compute node in the requested list of compute node host names. If the internal power state is standby or suspend, the script executes its IPMI `wake` command (whatever the admin configured in the script) that tells the node's BMC to bump the node into the active running state; otherwise, the script executes its IPMI `off` command (whatever the admin configured in the script) that tells the node's BMC to power on the node.

Some operating systems require the Wake-on-LAN bit to be enabled using a tool like *ethtool*. Also, Wake-on-LAN packets might be blocked by the router, but not always.

In this manner, the MWS RM Power Management plug-in queries the actual power state of individual compute nodes and returns to Moab the simple binary on/off state it understands for scheduling jobs to compute nodes. Likewise, Moab controls the actual power state of individual compute nodes using only its simple binary on/off command. This method of simple command and simple job-scheduling-ability state enables Moab to remain scalable and responsive for automatic power management control using green policies.

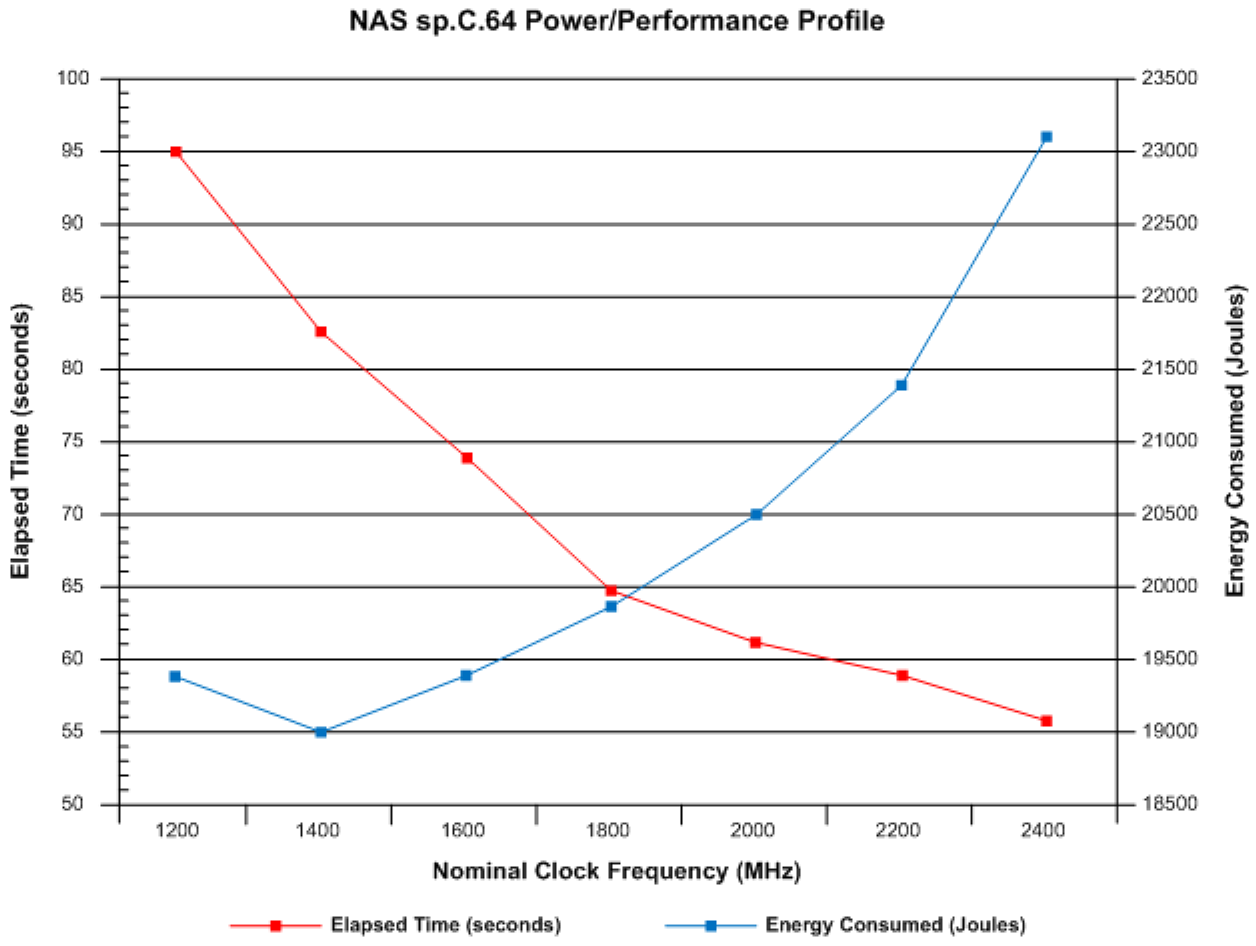
15.1.4 Active Node Power Management

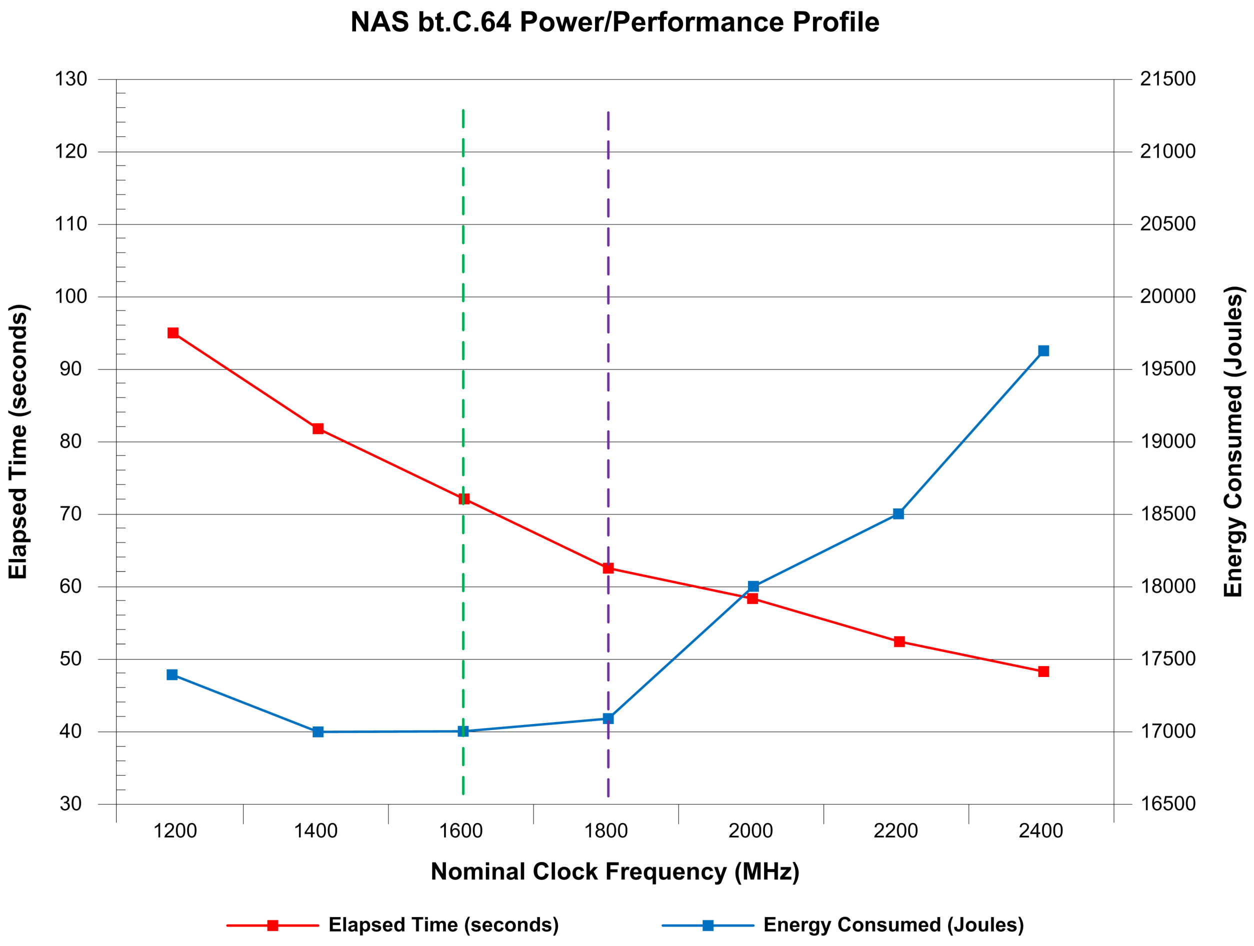
Moab and Torque have support for active node power management; that is, the management of energy consumption while a compute node is running a job, which the new CPU Clock Frequency Control feature provides.

The amount of energy consumption savings achievable through the CPU Clock Frequency Control feature is application-dependent. For example, memory, I/O, and/or network-bound applications, especially memory-bound applications, can often drop the clock frequency of their compute nodes' processors and still have the same execution time even though the compute nodes consume less power. Several studies have shown common power savings of 18-20% and one study showed one application saving 30% on power consumption, all of which translate directly into operational cost savings.

Power/Performance Profiling

To determine whether a lower clock frequency will produce energy consumption savings, applications must be profiled; that is, a job running a particular application with the same or equivalent data must be run at different clock frequencies while measuring the energy consumption of the job's compute node. Each pair of frequency/energy consumption data points are plotted in a chart to show the application's power performance profile. The charts below are an example of two such profiles for two NAS benchmark HPC applications.





The intersection of the two lines has no particular meaning, as each line has its own vertical scale, either on the left or the right as noted.

Note both applications do not consume the least energy (vertical dashed green line) when running at the lowest clock frequency, which demonstrates the importance of profiling applications to determine the nominal clock frequency at which energy consumption is the lowest. The charts amply illustrate why a simplistic policy of using the lowest clock frequency is not the best policy when a site's objective is the least energy consumption possible.

If the least energy consumption is not your primary objective, but running jobs in a manner that balances energy consumption and job execution time, a power/performance profile chart is very useful to determine the clock frequency that meets a balanced objective. For example, the vertical dashed purple line in the second chart shows that running the bt.C.64 application at 1800 MHz has an increase in energy consumption of ~1% over the minimal energy consumption possible (vertical dashed green line) but results in a ~10% drop in execution time; a possibly very good trade-off!

Obviously, if your primary objective is to complete a job as fast as possible but do so saving energy where possible, profiling memory-bound and other bound applications can clearly show the lowest clock frequency at which the application takes longer to execute. The site would then institute a policy that the application should run at the next highest frequency to fulfill the twin objectives of job performance and energy consumption minimization.

For more information about the CPU clock frequency job submission option, see the job extension CPUCLOCK of msub -l.

Job Templates

Most users will not care or want to know about clock frequency control, so admins can use a job template to specify the CPU clock frequency at which a particular recurring job should execute. A clock frequency specified on a job template overrides a clock frequency given on the job submission command line or inside a job script file with Torque PBS commands. This order of precedence enables an admin to control clock frequency for commonly used applications and jobs based on site policies and objectives.

For more information about using a CPU clock frequency job submission option in job templates, see the extension attribute CPUCLOCK.

15.1.5 Idle Node Power Management

Moab has so-called green policies that together configure Moab to manage and maintain a pool of idle nodes in an active running state so it can immediately schedule jobs to them. When Moab does so and diminishes the pool's idle compute node quantity, it powers on compute nodes by performing an `on` command for nodes in a powered-down state (actually, in a low-power or no-power state) to bring them on-line in order to replenish the pool of idle nodes up to its configured size. When jobs end and the idle node exceed the configured idle node pool size and there are no jobs to run on the now-idle nodes, Moab will power off excess idle nodes by performing an `off` command. In this manner, Moab achieves a site's power management and energy consumption objectives through the configured green policies.

See the Moab-only Method Architecture diagram above to see the color-coded compute nodes in the diagram's cluster illustrating Moab's green idle node pool management. The green nodes represent nodes running jobs, the yellow nodes are idle nodes in a green pool of size 12, and the gray nodes represent `off` nodes. Note Moab does not know what actual power state `off` means; what it means will be based on command customization inside Moab-only method scripts or Moab+MWS plug-in configuration information.

In order to perform green policy management of an idle node pool, Moab must first be configured to use either the Moab-only or the Moab+MWS method of power management. It is best practice to configure power management first and test its configuration before configuring green policies. Therefore, if power management is misconfigured, an admin will know it is the power management configuration and/or scripts and not the green computing policies that are incorrect. If the manual power management commands for the configured power management method work, green computing will work using the configured power management method. For information on how to configure each power management method in Moab, see 15.4 Enabling Green Computing.

15.1.6 Green Policy Configuration

There are several green policies that affect how Moab performs green idle node pool management using automated power management operations. The policies are configured in the same manner regardless of the power management method used, whether Moab-only or Moab+MWS. The other sections of this chapter describe how to configure green policies that manage the idle node pool for site energy management objectives.

Related Topics

- 15.2 Deploying Adaptive Computing IPMI Scripts
- 15.4 Enabling Green Computing
- pbsnodes in the *Torque Resource Manager Administrator Guide*

15.2 Deploying Adaptive Computing IPMI Scripts

If you want to enable green computing on your system using the Adaptive Computing supplied IPMI reference scripts, follow the steps here. The IPMI scripts provided are meant as a reference for you to configure the solution to your environment, but can also be used as-is.

In this section:

15.2.1 Prerequisites

- OpenIPMI and ipmitool must be installed and working.
- All nodes must have the same IPMI username and password.
- You must know the IPMI host names and/or IPMI IP addresses of your nodes.

- Python must be installed. The provided IPMI scripts were developed using Python 2.6.5.
- You must identify your Moab home directory. These instructions assume the default Moab home directory of `/opt/moab`.
- You must identify your Moab tools directory. These instructions assume the default Moab tools directory of `/opt/moab/tools`.

15.2.2 To Deploy the Adaptive Computing IPMI Scripts

1. Edit the `/opt/moab/tools/ipmi/config.py` script:
   a. Set `self.ipmiuser` to the IPMI username for your nodes.
   b. Set `self.ipmipass` to the location of the IPMI password file (`/opt/moab/passfile.txt` by default).

      The permissions for the directory and the password file itself should be set so that they can be read only by root or the Moab user running the script.

   c. Set `self.homeDir` to your Moab home directory.
   d. If desired, change the `self.pollInterval` value. This is the interval, in seconds, between polls from the IPMI monitoring script.
   e. The `self.ipmifile` value is the name of a temporary file where the cluster query information is stored. You can change this or leave it alone.
   f. The `self.bmcaddrmap` value is the filename for the Moab node name/IPMI mapping. The file must exist in the Moab home directory and will be created in the next step.
2. Create a `node-bmc.txt` file in the Moab home directory. The file must contain a space-delimited list of Moab node names that map to the IPMI host names or IP address. For Example:

```
node01 node01_ipmi   # For all three of these entries, the first value is the
node02 node02_ipmi   # node name as Moab knows it. The second value is either
node03 10.1.1.1      # the node IPMI name or IPMI IP address.
```

3. Configure the `moab.cfg` file for green computing as described in 15.4 Enabling Green Computing. Use the `ipmi.mon.py` script for the `CLUSTERQUERYURL` and the `ipmi.power.py` script for the `NODEPOWERURL`.
4. Restart Moab and verify green computing is working correctly. If you encounter trouble, see 15.9 Troubleshooting Green Computing topic for help.

Related Topics

- 15.4 Enabling Green Computing
- 15.5 Adjusting Green Pool Size
- 15.6 Handling Power-Related Events
- 15.7 Maximizing Scheduling Efficiency
- 15.9 Troubleshooting Green Computing

15.3 Choosing which Nodes Moab Powers On or Off

Moab can use the GREENPOOLPRIORITYF function to determine which nodes to power on or off. The PRIORITY node allocation policy is used to determine which nodes to allocate workload to. When Moab can no longer allocate workload to available nodes, it begins to power nodes on in the order specified by the GREENPOOLPRIORITYF function.

In this section:

15.3.1 To Choose which Nodes Moab Powers On or Off

Set a GREENPOOLPRIORITYF function to describe which order nodes should be selected for power on/off actions. GREENPOOLPRIORITYF uses the PRIORITY node allocation policy options and syntax.

```
GREENPOOLPRIORITYF '10*RANDOM'
```

This tells Moab to randomly choose a node to power on to meet workload demands, and to randomly choose an idle node to power off to meet the MAXGREENSTANDBYPOOLSIZE goal.

15.3.2 To Choose which Nodes Moab Allocates Jobs to

Set a PRIORITY node allocation policy that uses power as the major factor. This causes Moab to allocate jobs to nodes that are already powered on. When no nodes are available to meet this policy, Moab uses the GREENPOOLPRIORITYF function to turn on nodes that are powered off.

```
NODEALLOCATIONPOLICY  PRIORITY
NODECFG[DEFAULT] PRIORITYF='10000*POWER + 10*RANDOM'
```

The nodes with the highest priority for workload are the nodes that are powered on. After that, Moab randomly allocates workload.

Related Topics

- 15.5 Adjusting Green Pool Size
- 15.7 Maximizing Scheduling Efficiency

15.4 Enabling Green Computing

There are two ways to do green computing in Moab. With just Moab, nodes can be turned on or off. With MWS, however, you can put nodes into several low-power states. The MWS solution is also more scalable. The supported low-power states are:

- Running
- Standby
- Suspend
- Hibernate
- Shutdown

Nodes cannot be moved from one low-power state to another. The node must go from low-power to running, and then to the new low-power state.

In this section:

15.4.1 To Enable Green computing with Moab and MWS

1. Edit `moab.cfg` to use MWS for green computing:
   a. Configure the POWERPOLICY attribute of the NODECFG parameter. The default value is `STATIC`. Set it to `OnDemand`.
   b. Set the resource manager type as `MWS`.
   c. Set `FLAGS=UserSpaceIsSeparate` for the MWS resource manager.
   d. Point `BASEURL` to your MWS server.

```
NODECFG[DEFAULT]        POWERPOLICY=OnDemand
RMCFG[mws]              TYPE=MWS
RMCFG[mws]              FLAGS=UserSpaceIsSeparate
RMCFG[mws]              BASEURL=https://localhost:8080/mws
```

2. Configure the MWS Power Management plugin. See 'Power Management Plugin' in the Moab Web Services Administrator Guide.

15.4.2 To Enable Green Computing with just Moab

1. Edit `moab.cfg` to enable green computing. There are four things you must configure for basic functionality of green computing:
   a. Configure the POWERPOLICY attribute of the NODECFG parameter. The default value is `STATIC`. Set it to `OnDemand`.
   b. Configure a power provisioning resource manager to be TYPE=`NATIVE` and RESOURCETYPE=`PROV`. The resource type of `PROV` means the resource manager works only with node hardware and not workloads.
   c. Configure a CLUSTERQUERYURL attribute of the power provisioning resource manager to point to the power query script you'd like to use. Moab uses this script to query the current power state of the nodes. `CLUSTERQUERYURL` is traditionally used as a workload query but is also used by green computing for the node power state query. Adaptive Computing provides a reference IPMI script you can use.
   d. Configure a NODEPOWERURL attribute of the power provisioning resource manager to point to the power action script you'd like to use. Moab uses this script to turn nodes on or off. Adaptive Computing provides a reference IPMI script you can use.

```
NODECFG[DEFAULT] POWERPOLICY=OnDemand
RMCFG[ipmi] TYPE=NATIVE RESOURCETYPE=PROV
RMCFG[ipmi] CLUSTERQUERYURL=exec://$TOOLSDIR/ipmi/ipmi.mon.py
RMCFG[ipmi] NODEPOWERURL=exec://$TOOLSDIR/ipmi/ipmi.power.py
```

15.4.3 Sample moab.cfg for Green Computing

Below is a sample `moab.cfg` configuration file of a green computing setup using the Adaptive Computing IPMI scripts:

```
################################################################################
#
#  Use 'mdiag -C' to validate config file parameters
#
################################################################################

SCHEDCFG[Moab]        SERVER=myhostname:5150
ADMINCFG[1]           USERS=myusername,root
TOOLSDIR              /$HOME/tools
LOGLEVEL              1

################################################################################
#
#  Basic Resource Manager configuration
#
#  For more information on configuring a Resource Manager, see:
#  docs.adaptivecomputing.com
#
################################################################################

RMCFG[local]   TYPE=NATIVE
RMCFG[local]   CLUSTERQUERYURL=exec://$HOME/scripts/query.resource
RMCFG[local]   WORKLOADQUERYURL=exec://$HOME/scripts/query.workload

RMCFG[local]   JOBSUBMITURL=exec://$HOME/scripts/submit.pl
RMCFG[local]   JOBSTARTURL=exec://$HOME/scripts/job.start
RMCFG[local]   JOBCANCELURL=exec://$HOME/scripts/job.cancel
RMCFG[local]   JOBMODIFYURL=exec://$HOME/scripts/job.modify
RMCFG[local]   JOBREQUEUEURL=exec://$HOME/scripts/job.requeue
RMCFG[local]   JOBSUSPENDURL=exec://$HOME/scripts/job.suspend
RMCFG[local]   JOBRESUMEURL=exec://$HOME/scripts/job.resume

##################################
# GREEN configuration:
##################################
# Turn on "green" policy. (This is the policy that enables green computing).
# Here we are doing it for all nodes, but it can be controlled on a node-by-node basis
# Default is STATIC, which means green computing is disabled.
#NODECFG[DEFAULT]  POWERPOLICY=STATIC
NODECFG[DEFAULT]  POWERPOLICY=OnDemand

# Configure the power provisioning and power state query scripts for the power
# management system.
# Note that this is an entirely different RM (with a name of power in this case
# and a type of 'PROV').
# The PROV type RM is the only one that uses a NODEPOWERURL.  Additionally, the
# output of the CLUSTERQUERYURL for this type of RM is different.  (See docs)
RMCFG[mws]  TYPE=MWS
RMCFG[mws]  FLAGS=UserSpaceIsSeparate
RMCFG[mws]  BASEURL=https://localhost:8080/mws

# We want green policy to work so it allocates jobs to compute nodes already
# powered on and will power on powered-off compute nodes only when there are
# no powered-on compute nodes available. This requires using the PRIORITY
# node allocation policy with a PRIORITYF function that has the POWER variable
# as the greatest contributing factor to the function (1 = powered-on,
# 0 = powered-off).
# If we want all compute nodes to operate under green policy, we can assign
# the PRIORITYF function to the default node configuration, which is easier
# than assigning it to individual compute nodes. If only some compute nodes
# should operate under green policy, then the PRIORITYF function must be
# configured for the individual nodes. Note the POWER variable must be the
# largest factor in the function below; it is assigned the largest multiplier,
# which should be greater than the sum of all other factors! Doing so forces
# Moab to use all eligible powered-on nodes for workload placement before
# powering on any eligible powered-off nodes.

# Enable PRIORITYF functionality
NODEALLOCATIONPOLICY  PRIORITY

# Use a priority function that uses power as the major factor (plus some other
imaginary factors)
#NODECFG[DEFAULT]      PRIORITYF='1000000*POWER + 1000*factor2 + 100*factor3...'
# Use a priority function where power is the only factor.
#NODECFG[DEFAULT]      PRIORITYF='10000*POWER'
# Use a priority function that adds some randomness but uses power as the major
factor.
NODECFG[DEFAULT]      PRIORITYF='10000*POWER + 10*RANDOM'

# Set a priority function that specifies the order nodes should be chosen to power
# up/down. By default, Moab will start at the top of the node list and go down. Some
# installations want to rotate power cycles among nodes in a different order.
# The configuration below forces Moab to power on/off random nodes, which
# eventually guarantees all nodes occasionally go through a power cycle.
#GREENPOOLPRIORITYF '10*RANDOM'

# Ensure we are recording power management events
# (powering on and off nodes are recorded as "node modification" events).
#RECORDEVENTLIST +NODEMODIFY

# Set the size of the standby pool. This is the number of idle nodes that will
# be powered on and idle. As the workload changes, Moab turns nodes on
# or off to try to meet this goal.
# Default value is -1 (all idle nodes are left powered on by default)
MAXGREENSTANDBYPOOLSIZE 5

# Set the length of time that it takes to power a node on/off. This will be the
# walltime of the system job that performs the power operation and should be the
# maximum expected time. If Moab detects (via the power RM) that the power
# operations have all completed, the system job will finish early.
# Default value is 10 minutes (600)
PARCFG[ALL]       NODEPOWEROFFDURATION=600
PARCFG[ALL]       NODEPOWERONDURATION=600
# Set the length of time a node should remain idle before it is powered off.
# This prevents Moab from immediately powering off nodes that have just finished
# a job. Increasing this number should decrease power on/off thrashing
# This should be set higher than NODEPOWEROFFDURATION and/or NODEPOWERONDURATION
 NODEIDLEPOWERTHRESHOLD 660

# If a node fails to power on, we need to remove it from the available nodes so
# Moab won't keep [re-]trying to power it on. Do this by setting a reservation
# on the failed node to give time for manual investigation.
#RMCFG[torque] NODEFAILURERSVPROFILE=failure
#RSVPROFILE[failure] DURATION=3600
```

Related Topics

- 15.2 Deploying Adaptive Computing IPMI Scripts
- 15.3 Choosing which Nodes Moab Powers On or Off
- 15.5 Adjusting Green Pool Size
- 15.6 Handling Power-Related Events
- 15.7 Maximizing Scheduling Efficiency
- 15.9 Troubleshooting Green Computing
- Power Management Plugin in the *Moab Web Services Administrator Guide*

15.5 Adjusting Green Pool Size

The MAXGREENSTANDBYPOOLSIZE parameter enables you to allocate the number of nodes to keep powered on in the standby pool. This is the number of idle nodes that are allowed to be powered on and idle. As the workload changes, Moab turns nodes on or off to try to meet this goal. The default value is -1, which disables the standby pool.

To Adjust the Green Pool Size

Modify the MAXGREENSTANDBYPOOLSIZE parameter with the number of nodes you want Moab to keep powered on for the standby pool:

```
MAXGREENSTANDBYPOOLSIZE 10
```

Moab keeps up to 10 idle nodes powered on to be kept on standby.

Related Topics

- 15.3 Choosing which Nodes Moab Powers On or Off
- 15.7 Maximizing Scheduling Efficiency

15.6 Handling Power-Related Events

Power actions are considered NODEMODIFYURL events and are not recorded by default, but you can configure Moab to include power-related events in the logs. Also, if a node fails to turn on (or off), it's best to associate a reservation on the failed node so that Moab won't keep trying to perform the power action over and over.

To Configure Moab to Record Power-Related Events

Modify the RECORDEVENTLIST parameter:

```
RECORDEVENTLIST +NODEMODIFY
```

Power-related events are logged to the Moab log file.

To Put a Reservation on a Node that Fails to Perform a Power Action

Configure the NODEFAILURERSVPROFILE attribute of RMCFG and create an RSVPROFILE with a high duration:

```
RMCFG[torque] NODEFAILURERSVPROFILE=failure
RSVPROFILE[failure] DURATION=3600
```

Nodes that fail to power on or off have a 1-hour reservation placed on them.

Related Topics

- Appendix A: Moab Parameters - RECORDEVENTLIST
- 12.2.4 Event Logs

15.7 Maximizing Scheduling Efficiency

When considering whether to power a node on or off, Moab can take into account the amount of time that it takes to power on or power off the node. With this information, Moab can keep an idle node powered on if it knows that workload in the queue will be ready for the node in less time that it takes to power off/power on the node.

Moab can also wait to shut down nodes after they've been idle for a specific amount of time.

To Specify Node Power On/Power Off Duration

Modify the NODEPOWERONDURATION and NODEPOWEROFFDURATION attributes of PARCFG with the maximum amount of time it takes for your nodes to power on/power off. Make sure to use the keyword ALL for the resource manager name to avoid cases where Moab won't consider the power on/off duration for a node before making a power action decision.

```
PARCFG[ALL] NODEPOWERONDURATION=2:00
PARCFG[ALL] NODEPOWEROFFDURATION=2:00
```

If a node goes idle and has to wait for workload, Moab will not power off the node if the workload will be available within 4 minutes or less.

To Shut Down on Nodes after they've been Idle for a Specified Time

Modify the NODEIDLEPOWERTHRESHOLD parameter with the duration (in seconds) you want Moab to wait before shutting down an idle node. The default value is 60 seconds. Increasing the number should decrease power on/off thrashing. This should be set higher than NODEPOWERONDURATION and/or NODEPOWEROFFDURATION.

```
NODEIDLEPOWERTHRESHOLD 300
```

Moab will wait 5 minutes before shutting down a node that has become idle.

Related Topics

- 15.3 Choosing which Nodes Moab Powers On or Off
- 15.5 Adjusting Green Pool Size

15.8 Putting Idle Nodes in Power-Saving States

When nodes exceed their idle threshold limits, the default behavior is to turn the nodes off. With the NODEIDLEPOWERACTION parameter, you can choose which power-saving state to put idle nodes into. This parameter is configured at the partition level. Configuring it for the ALL partition effectively makes it a global parameter.

To Specify what to do with Idle Nodes

Modify the NODEIDLEPOWERACTION parameter:

```
NODEIDLEPOWERTHRESHOLD 300
PARCFG[ALL] NODEIDLEPOWERACTION SLEEP
```

All nodes that are idle for more than 5 minutes are put into a sleep state.

Related Topics

- 15.3 Choosing which Nodes Moab Powers On or Off

15.9 Troubleshooting Green Computing

If you've enabled green computing and are having trouble, here are some tips that can help you determine the cause of the issues you encounter. These tips are specifically for the Adaptive Computing supplied IPMI scripts, but can be generalized for whatever power management solution you use. Simply substitute your power management system, power query script (as specified by `CLUSTERQUERYURL`), and power action script (as specified by `NODEPOWERURL`) where appropriate.

Verify your IPMI Access

Use the *`ipmitool`* command to verify you have access to the IPMI interface of your nodes. Try getting the current power state of a node. The syntax is `ipmitool -I lan -H <host> -U <IPMI username> -P <IPMI password> chassis power status`.

```
$ ipmitool -I lan -H qt06 -U ADMIN -P ADMIN chassis power status

Chassis Power is off
```

Verify the Power Query (CLUSTERQUERYURL) Script is Working

1. Execute the `impi.mon.py` script (should be found in `/<MOABHOMEDIR>/tools/ipmi`) to start the monitor:

```
$ cd /opt/moab/tools/ipmi
$ ./ipmi.mon.py
```

2. Execute the script again. The following is an example of the expected output:

```
$ ./ipmi.mon.py

qt09  GMETRIC[System_Temp]=27 GMETRIC[CPU_Temp]=25 POWER=on State=Unknown
qt08  GMETRIC[System_Temp]=31 GMETRIC[CPU_Temp]=25 POWER=on State=Unknown
qt07  GMETRIC[System_Temp]=30 GMETRIC[CPU_Temp]=29 POWER=on State=Unknown
qt06  GMETRIC[System_Temp]=Disabled GMETRIC[CPU_Temp]=Disabled POWER=off
State=Unknown
```

If the **POWER** attribute is not present, the script is not working correctly.

Verify the Power Action (NODEPOWERURL) Script is Working

1. Execute the `ipmi.power.py` script (should be found in `/<MOABHOMEDIR>/tools/ipmi`) to see if you can force a node to power on or off. The syntax is `ipmi.power.py <node>,<node>,<node>... [off|on]`.

```
$ /opt/moab/tools/ipmi/ipmi.power.py qt06 off
```

This example is trying to power off a node named qt06.

2. Verify the machine's power state was changed to what you attempted in the previous step. You can do this remotely via two methods:
   a. If the cluster query script is working, you can use that to verify the current power state of the node.
   b. If you have IPMI access, you can use the *ipmitool* command to verify the current power state of the node.

Verify the Scripts are Configured Correctly

1. Run the mdiag -R command to verify your IPMI resource manager configuration:

```
$ mdiag -R -v
RM[ipmi]      State: Active  Type: NATIVE  ResourceType: PROV
Timeout:             30000.00 ms
Cluster Query URL:   exec://$TOOLSDIR/ipmi/ipmi.mon.py
Node Power URL:      exec://$TOOLSDIR/ipmi/ipmi.power.py
Objects Reported:    Nodes=3 (0 procs)  Jobs=0
Nodes Reported:      3 (N/A)
Partition:           SHARED
Event Management:    (event interface disabled)
RM Performance:      AvgTime=0.05s  MaxTime=0.06s  (176 samples)
RM Languages:        NATIVE
RM Sub-Languages:    NATIVE
```

2. Run the *mdiag -G* command to verify that power information is being reported correctly:

```
$ mdiag -G

NodeID      State      Power   Watts   PWatts
 qt09       Idle       On      0.00    0.00
 qt08       Idle       On      0.00    0.00
 qt07       Idle       Off     0.00    0.00
```

Verify the Scripts are Running

Once green is configured and Moab is running, Moab should start the power query script automatically. Use the *ps* command to verify the script is running:

```
$ ps -ef | grep <CLUSTERQUERYURL script name>
```

If this command does not show the power query script running then your settings in `moab.cfg` aren't working.

Verify Moab can Power Nodes On or Off

Use the mnodectl command to turn a node on or off. The syntax is `mnodectl -m power=[off|on] <node>`.

```
mnodectl -m power=off qt06
```

Moab should turn off the node named qt06.

Moab generates a system job called `poweron-<num>` or `poweroff-<num>` job as shown in showq. The system job calls the `ipmi.power.py` (`NODEPOWERURL`) script to execute the command.

Moab waits until the cluster query reports the correct data. In this case, the `ipmi.power.py` script reports that the power attribute has changed.

Moab does not change the power status based on the power script return code. Rather, Moab completes the system power job when it detects the power attribute has changed as indicated by the cluster query script.

Related Topics

- 15.2 Deploying Adaptive Computing IPMI Scripts
- 15.4 Enabling Green Computing

Chapter 16: Elastic Computing

Elastic Computing is an add-on package for Moab. Contact your Adaptive Computing account manager for more information.

Elastic Computing is only applicable for Torque Resource Manager and Native RMs with QoS triggers.

Elastic Computing is not supported on Ubuntu.

During the course of operation, the number of job requests will go up and down. Under some circumstances the job backlog might increase to the point where additional resource are required to complete the job backlog in a reasonable time frame. In this scenario, the job will be held until resources become available. The Elastic Computing feature in Moab enables the Moab scheduler to take advantage of systems that can temporarily provide additional nodes (for example, to create new virtual machines or borrow physical nodes from another system) to fulfill the workload demand in a more timely manner.

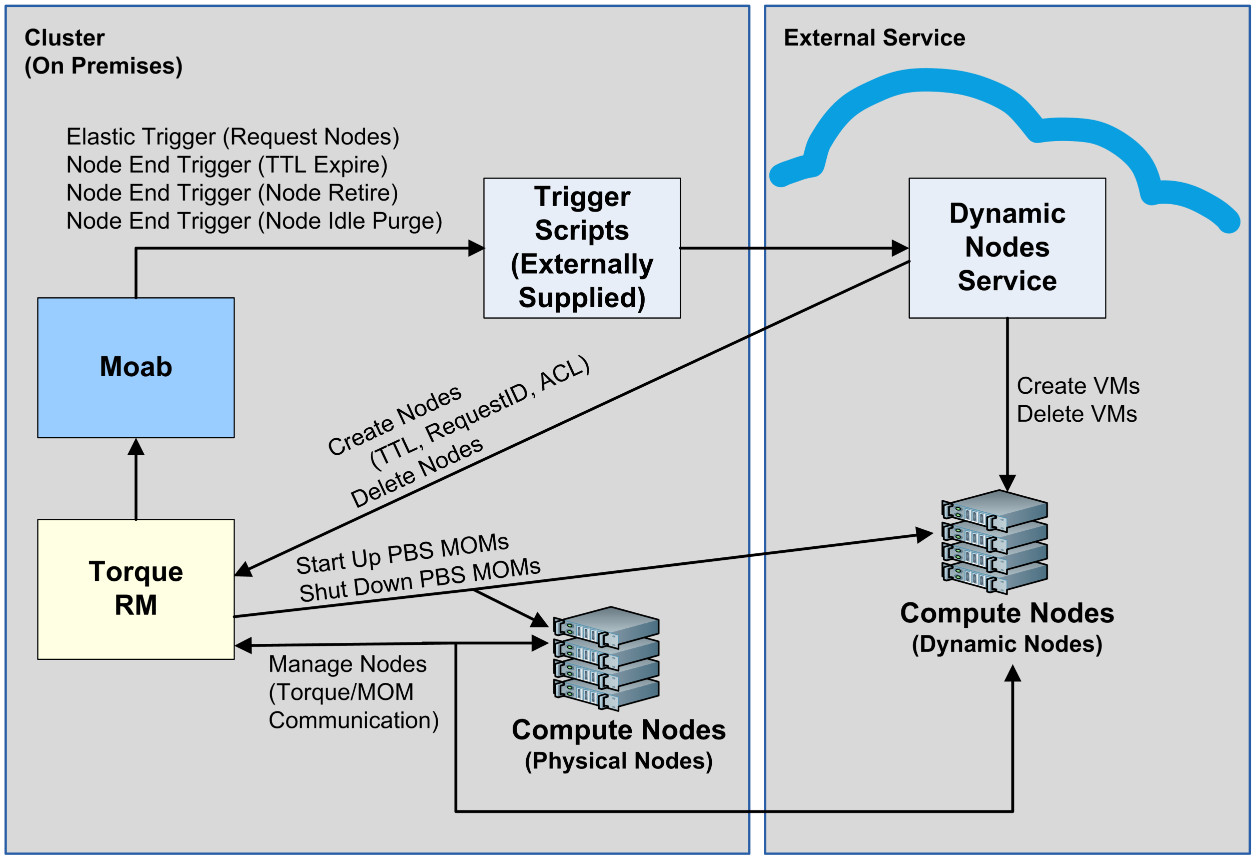
Moab's Elastic Computing framework serves as a basis for Moab/ODDC Cloud Bursting, which can be configured to access multiple cloud providers either on-demand or based on a job backlog.

This chapter provides examples of the Elastic Computing and node end scripts. Your scripts will vary based on your system configuration. Contact your Adaptive Computing account manager for suggestions and options to configure Elastic Computing.

In this chapter:

16.1 Elastic Computing

The diagram below depicts Moab's Elastic Computing feature:



With the Elastic Computing feature enabled and configured:

1. Moab monitors the job backlog and, when a predefined threshold is reached, fires the elastic trigger. The elastic trigger calls a script to request additional nodes (dynamic nodes) from an external service.
2. The procured dynamic nodes are then added to the resource manager. For example, via *qmgr* in Torque.
3. Moab then begins scheduling jobs for the allocated dynamic nodes.
4. When the job backlog is cleared and the dynamic nodes become idle for a specified amount of time (for example, defined by the NODEIDLEPURGETIME parameter), Moab fires an end node trigger to remove the nodes from the resource manager and deprovision the virtual machine or physical nodes.

Alternatively, you can utilize the Elastic Computing feature for interaction with a private OpenStack cloud. See 16.4 Integration with a Private OpenStack Cloud.

16.2 Configuring Elastic Computing

This section provides examples of the Elastic Computing and node end scripts. Your scripts will vary based on your system configuration. Contact your Adaptive Computing account manager for suggestions and options to configure Elastic Computing.

If you are using Elastic Computing with Torque, you cannot have a `mom_hierarchy` file in the `$PBS_HOME/server_priv` directory.

In this section:

16.2.1 To Configure Elastic Computing

1. If you installed Moab from the tarball (Manual Installation), ensure you installed acpython-base RPM on the Moab Head Node:

   ```
   [root]# rpm -qa|grep acpython-base
   ```

   If it is not, follow the instructions in 'Install Moab Server' in the *Moab HPC Suite Installation and Configuration Guide*.

2. Enable dynamic nodes in the `moab.cfg` file:

   ```
   SCHEDCFG[] FLAGS=enabledynamicnodes
   ```

   A sample excerpt from a `moab.cfg` file is shown in Sample moab.cfg File Excerpt below.

3. If you want to be able to view node and trigger information, use one of these Moab tools:

   - `mdiag -n -v -xml`
   - `mdiag -T`
   - `checknode -v <node name>`

   See 16.6 Viewing Node and Trigger Information for more information.

4. If you want to record dynamic node activity, enable NODEADD and/or NODEREMOVE for RECORDEVENTLIST in the `moag.cfg` file. See 16.5 Dynamic Nodes for more information.
5. In the `moab.cfg` file, make these changes for QoS triggers:
   a. Add the elastic trigger: `TType=elastic`. See 16.3 Elastic Trigger for more information.
   b. Specify how nodes are requested when the trigger fires, using one of these two options:

      -
        ```
        QOSCFG[xyz] REQUESTGEOMETRY=12@4:00:00:00
        ```
        When the elastic trigger fires, request 12 additional nodes for 4 days, 0 hours, 0 minutes, and 0 seconds.

        The `REQUESTGEOMETRY` values shown are just an example.

      -
        ```
        QOSCFG[xyz] REQUESTGEOMETRY=PRIORITYJOBSIZE
        ```
        When the elastic trigger fires, request enough nodes to run the highest priority job in the backlog for the amount of walltime specified by the highest priority job.
6. Use the metric BACKLOGCOMPLETIONTIME to specify when the elastic trigger fires (adding nodes).

   The trigger that contains the `BACKLOGCOMPLETIONTIME` threshold can only be used when profiling is enabled.

   The `BACKLOGCOMPLETIONTIME` is calculated by Moab as follows: (the maximum number of processor seconds in the QoS) divided by (the total number of processors in the system).

   ```
   NODECFG[DEFAULT] ENABLEPROFILING=TRUE
   QOSCFG[xyz] TRIGGER=EType=threshold,AType=exec,Action="$HOME/tools/elastic.py
   $REQUESTGEOMETRY",Threshold=BACKLOGCOMPLETIONTIME>1800,RearmTime=05:00
   ```

   In the above example, when the `BACKLOGCOMPLETIONTIME` is more than 1800 seconds, the `QOSCFG` threshold trigger will fire. When the `QOSCFG` trigger is fired, the `$HOME/tools/elastic.py` script is executed. This is a user-supplied script that needs to create virtual machines or provision physical hardware and add these dynamic nodes to the resource manager.

   The following examples show commands that the script will run in order to create a node on Torque:

```
qmgr -c "create node node01 np=4,TTL=2024-09-
26T12:00:00Z,acl='user==user1',requestid=1234"
```

Once the BACKLOGCOMPLETIONTIME threshold is reached, the trigger will begin firing. The admin can configure the trigger to fire once only or periodically until the node is deleted from Torque by the external service.

7. Determine how the dynamic nodes will be removed. See 16.5 Dynamic Nodes for more information.

   a. Use one or both of these methods:

   - Set the TTL when creating the node via the resource manager. This parameter tells Moab to remove the node when the TTL has passed.
   - Add the NODEIDLEPURGETIME parameter to moab.cfg. To turn off the purging of *individual* dynamic nodes output, specify "noidlepurge" in the varattr output of the node using Torque. See '$varattr' in the *Torque Resource Manager Administrator Guide*. Alternatively, you can use the varattr output from the wiki interface. See the query resource VARATTR.

   You can optionally report a requestid on each node in the same group.

   Nodes without a requestid that hit the configured idle purge time are immediately purged. Whereas, nodes with a requestid that hit the configured idle purge time are only purged when all the nodes that have the same requestid hit the configured idle purge time.

   b. Configure the node end trigger in moab.cfg:

```
NODECFG[DEFAULT]
TRIGGER=EType=end,TType=elastic,AType=exec,Action="/$HOME/tools/nodeend.sh $OID"
```

   In this example, the nodeend.sh trigger will be called with the name of each node in the requestid group.

   The node end trigger notifies the external service that this node (along with all the other nodes with the same requestid) has met the node idle purge time. The external service may then choose to remove the node from Torque (which in turn removes it from Moab).

   The following is an example of the command that a service runs to remove a node from Torque:

```
qmgr -c 'delete node node01'
```

8. If you want to set limits on whether bursting is available, specify the limits using the usage policies. You can set these limits at the global partition or QoS level. See 16.7 Usage Policies.

16.2.2 Sample moab.cfg File Excerpt

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
SCHEDCFG[moab] FLAGS=enabledynamicnodes
QOSCFG[xyz] REQUESTGEOMETRY=12@4:00:00:00
QOSCFG[xyz] TRIGGER=EType=threshold,AType=exec,Action="$HOME/tools/elastic.py
$REQUESTGEOMETRY",Threshold=BACKLOGCOMPLETIONTIME>1800,RearmTime=05:00
NODEIDLEPURGETIME 3600
NODECFG[DEFAULT]
TRIGGER=EType=end,TType=elastic,AType=exec,Action="/$HOME/tools/nodeend.sh $OID"
```

16.3 Elastic Trigger

When enabled, the elastic trigger enables the Moab scheduler to take advantage of systems that can temporarily provide additional nodes to fulfill the backlog in a reasonable time frame.

The elastic trigger is added to `moab.cfg` when the TType trigger component is set to 'elastic'. See 17.3.7 Trigger Components for more information.

When configured and enabled, this trigger:

- Takes the `REQUESTGEOMETRY` parameter and creates nodes in provider.
- Adds nodes to Torque using the create node qmgr command (or optionally add it to their resource manager's cluster query).
- Makes sure that `TTL` is set correctly on the new nodes.
- Optionally adds a request ID (generated by the script) and/or ACL to the nodes.

Elastic Computing scripts should only service one request at a time, we recommend to not return until the node is up and free in the resource manager.

Example

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
QOSCFG[xyz]
TRIGGER=EType=threshold,AType=exec,TType=elastic,Action="$HOME/tools/elastic.py
$REQUESTGEOMETRY",Threshold=BACKLOGCOMPLETIONTIME>1800,RearmTime=05:00
```

The BACKLOGCOMPLETION time trigger threshold can only be used when profiling is enabled.

16.4 Integration with a Private OpenStack Cloud

Adaptive Computing has provided a services-enabled integration with a private OpenStack cloud. This consists of the MWS OpenStack plug-in, integration scripts, and Moab configuration.

This section provides instructions to configure your system to send elastic compute requests to OpenStack.

In this section:

Parameter	Description
OpenStack Customization Script (cloudinit)	This can be used to perform post-creation actions on the provisioned machines.
OpenStack VLAN Name	Use this if the correct IP address is not returned from the OpenStack plug-in after provisioning a machine.
OpenStack Keypair Name	Use this parameter to gain access to the machine after provisioning.

16.4.1 Configuring Moab to Talk to OpenStack Integration Scripts

The following configuration snippet shows how to configure Moab to use the OpenStack plug-in:

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
SCHEDCFG[moab] FLAGS=enabledynamicnodes
QOSCFG[xyz] REQUESTGEOMETRY=12@4:00:00:00

QOSCFG[xyz]
```

```
TRIGGER=EType=threshold,AType=exec,TType=elastic,Action="$HOME/tools/openstack/opensta
ck_elastic.py $REQUESTGEOMETRY",Threshold=BACKLOGCOMPLETIONTIME>1800,RearmTime=05:00
NODEIDLEPURGETIME 3600
NODECFG[DEFAULT]
TRIGGER=EType=end,TType=elastic,AType=exec,Action="/$HOME/tools/openstack/openstack_
delete.py $OID"
```

See 16.2 Configuring Elastic Computing for more information on how to configure elastic triggers and thresholds.

These options are also available on the OpenStack elastic script to fine-tune the configuration:

Option	Sample Value	Description
--acl	user=bob	Set ACLs on the provisioned machines to a specific value.
--ttl-pad	360	By default, the requested TTL is 'padded' by 3 minutes to allow for provisioning time. This parameter can be used to increase or decrease the padding time.

For example, to set an ACL and increase the TTL padding to 4 minutes, use the following elastic trigger definition:

```
QOSCFG[xyz]
TRIGGER=EType=threshold,AType=exec,TType=elastic,Action="$HOME/tools/openstack/opensta
ck_elastic.py --acl=user=alice --ttl-pad=240
$REQUESTGEOMETRY",Threshold=BACKLOGCOMPLETIONTIME>1800,RearmTime=05:00
```

16.4.2 Verification

The following methods can be used to verify that the configuration is correct:

- The triggerElastic and triggerNodeEnd web services can be called directly through a browser or command line utility to ensure that the plug-in can interact with OpenStack correctly.
- The openstack_elastic.py and openstack_delete.py scripts can be called directly through the command line. Make sure to match the parameters used in your Moab configuration file.
- Submit enough workload to cause Moab to fire the elastic trigger and observe that the process works end-to-end.

16.4.3 Troubleshooting

The following methods can be used to troubleshoot the OpenStack integration:

- Check the output of `mdiag -T` for information on the configured triggers and to see whether they are firing and results of their execution.
- Check the trigger script log files located in the Moab log directory. By default, these are located at `/opt/moab/log/openstack_elastic.log` and `opt/moab/log/openstack_delete.log`.
- Check the MWS log for information on provisioning and deleting OpenStack machines.
- If the triggers are not firing, check the Moab Workload Manager log files for additional information.

16.5 Dynamic Nodes

Dynamic nodes are nodes that can be added and removed from Torque at any time. Specifically, any node that has a TTL (time to live) is considered a dynamic node. The following section explains how to add and delete nodes via `qmgr`.

As of Moab version 9.1.2, dynamic node procs are no longer counted against the total procs listed in the Moab license. This enables you to do as many bursts as you desire without exceeding the total procs used for on-premises nodes. If your version of Moab is before 9.1.2, contact your Adaptive Computing sales representative.

In this section:

16.5.1 Dynamic Node Parameters

The table below describes the parameters that are used while adding and removing dynamic nodes:

Parameter Name	Required/ Optional	Data Format	Description
TTL	Optional	yyyy-mm-ddThh:mm:ss±hh OR yyyy-mm-ddThh:mm:ss±hhmm OR yyyy-mm-ddThh:mm:ssZ	Time, given as a UTC time, for the node to be removed. The time is Greenwich Mean Time with either an offset or a Z to indicate zero offset.
requestid	Optional	Any sequence of non-white-space characters	Identifier used by Moab to identify a group of nodes. See the section requestid Parameter (Adding or Removing Nodes) below for more information.
acl	Optional	user==user1:user2,host==host1	List of credentials that can run jobs on this dynamic node.

16.5.2 Dynamic Node Events

You can record dynamic node activity using RECORDEVENTLIST in the `moab.cfg` using one or both of these events:

- NODEADD
- NODEREMOVE

16.5.2.A NODEADD

The `NODEADD` event is generated when the resource manager first reports a new node to Moab.

The following is an example from the `event_xxx` file in the `$MOAB_HOME/stats` directory:

```
16:22:32 1412202152:359437 node     nuc2          NODEADD      nuc2 STATE=Idle
PARTITION=bdaw ADISK=1 AMEMORY=15193 APROC=4 ASWAP=16717 CDISK=1 CMEMORY=15918 CPROC=4
CSWAP=17442 OS=linux RM=bdaw NODEACCESSPOLICY=SHARED CCLASS=[DevQ][batch] MSG='Node
'nuc2' was newly reported in the last cluster query.  RequestID = 1234, TTL =
1420070400'
```

16.5.2.B NODEREMOVE

The `NODEREMOVE` event is generated when Moab removes a dynamic node after `TTL` has expired, or if the node is no longer reported to Moab by the resource manager.

The following is an example from the `event_xxx` file in the `$MOAB_HOME/stats` directory:

```
16:21:44 1412202104:359401 node     nuc2         NODEREMOVE   nuc2 STATE=Idle
PARTITION=bdaw ADISK=1 AMEMORY=15192 APROC=4 ASWAP=16716 CDISK=1 CMEMORY=15918 CPROC=4
CSWAP=17442 OS=linux RM=bdaw NODEACCESSPOLICY=SHARED FEATURE=[DEV] CCLASS=[DevQ]
[batch] MSG='Dynamic node 'nuc2' is being removed.  RequestID = 1234, TTL =
1420070400, Reason = node removed because the RM did not report it in the cluster
query'
```

16.5.3 Configuring Dynamic Nodes

This section contains information on configuration options when adding or removing nodes:

- TTL Parameter (Creating Nodes)
- requestid Parameter (Adding or Removing Nodes)
- NODEIDLEPURGETIME Parameter (Removing Nodes)

During the creation of a dynamic node, the pbs_server will attempt to resolve the node name to an IP address. If pbs_server is unable to resolve the name, it will not create the node; nor will it retry the creation later.

Immediately after a dynamic node is created, it is assigned a state of 'down|MOM-list-not-sent'. Once the new node has received the list of all moms, it will be assigned a state of 'free' and be available for job scheduling.

16.5.3.A TTL Parameter (Creating Nodes)

The dynamic nodes are added to the resource manager with a `TTL` parameter. The `TTL` parameter is passed to Moab by the resource manager. Moab does not schedule workload for a node beyond the `TTL` assigned to it. Moab removes a dynamic node when it reaches its expiration date as set by `TTL`. A node end trigger will then fire to notify the service that the dynamic node has been removed in Moab and the service may destroy the virtual machine or deprovision the physical nodes at its convenience.

The following is an example of a node being created with a `TTL` parameter:

```
qmgr -c 'create node node003[,node004,node005...] [np=n,][TTL=2024-05-16T05:26:30Z,]
[acl="user==user1:user2:user3",][requestid=n]'
```

In the above example, `node003` is created with `TTL=2024-05-16T05:26:30Z` as the `TTL` parameter. The dynamic node will be removed when the `TTL` is expired.

16.5.3.B requestid Parameter (Adding or Removing Nodes)

The dynamic nodes are added to the resource manager with a `requestid` parameter that is passed to Moab by the resource manager. Moab reports the `requestid` parameter along with the node ID in Moab logs, events, and node end triggers. This enables the external service to tag the nodes allocated together in a block. The tagged nodes are then associated as events, and are reported on a node-by-node basis by Moab.

The `requestid` can also be used by the external service to de-allocate nodes together in the same block as they were created by the service. For example, a group of nodes has their node end trigger fired due to node idle purge time or `TTL` expiration.

The `requestid` is useful if nodes are dynamically added, removed, and then re-added at some later time with the same node ID. Using a `requestid` when a node is re-added, will help identify each unique instance of a dynamic node's lifetime in logs, events, etc.

Moab also uses the `requestid` with the `NODEIDLEPURGETIME` parameter. The `requestid` parameter groups the nodes and then references the `NODEIDLEPURGETIME` information, if specified, to determine when to remove the group of nodes. When all the nodes associated with the `requestid` have reached the idle purge time threshold defined by the `NODEIDLEPURGETIME` parameter, Moab fires the node end trigger for all the nodes with the same `requestid`.

When requestid is configured with `NODEIDLEPURGETIME`, *all* of the nodes must be idle.

16.5.3.C NODEIDLEPURGETIME Parameter (Removing Nodes)

The `NODEIDLEPURGETIME` parameter instructs Moab to fire a node end trigger when all the nodes in the `requestid` group have been idle for the time period specified by `NODEIDLEPURGETIME`.

Setting the `NODEIDLEPURGETIME` to 0 effectively disables the `NODEIDLEPURGETIME`. The default value is 0 if `NODEIDLEPURGETIME` is not configured in the `moab.cfg` file. See the parameter `NODEIDLEPURGETIME` for more information.

The following is an example of configuring the node end trigger in `moab.cfg`:

```
NODECFG[DEFAULT]
TRIGGER=EType=end,TType=elastic,AType=exec,Action="/$HOME/tools/nodeend.sh $OID"
```

In this example, the `nodeend.sh` trigger will be called with the name of each node in the `requestid` group.

The node end trigger notifies the external service that the node (along with all the other nodes with the same `requestid`) has met the node idle purge time set by the `NODEIDLEPURGETIME` parameter. The external service may then choose to remove the node from Torque (which in turn removes it from Moab).

The following is an example of the command that a service will run to remove a node from Torque:

```
qmgr -c 'delete node node003'
```

If a job is running on a node when it is deleted, the job will be requeued if the job is requeueable or deleted if it is not. If the node has already been shut down, any jobs running on the node will be immediately purged.

16.6 Viewing Node and Trigger Information

You can optionally configure Elastic Computing to allow you to view the node and trigger information using the Moab commands described in this topic.

In this section:

16.6.1 mdiag -n -v --xml

The *`mdiag -n -v --xml`* command provides detailed information about the state of nodes that Moab is currently tracking. See `3.7.10 mdiag -n` for more information.

In the following example, the *`mdiag -n - v --xml`* command shows the current list of nodes including dynamic node parameters `TTL` and `REQUESTID` in the XML format:

```
$ mdiag -n -v --xml | xmllint --format -
<?xml version="1.0"?>
<Data>
  <node ACL="USER=%=bdaw+:%=adaptive+;" AVLCLASS="[DevQ][batch]" CFGCLASS="[DevQ]
[batch]" FEATURES="DEV" LASTUPDATETIME="1412200545" LOAD="0.330000" MAXJOB="0"
MAXJOBPERUSER="0" MAXLOAD="0.000000" NODEID="bdaw" NODEINDEX="0" NODESTATE="Idle"
OS="linux" OSLIST="linux" PARTITION="bdaw" PRIORITY="0" PROCSPEED="0" RADISK="1"
```

```
RAMEM="9746" RAPROC="1" RASWAP="26128" RCDISK="1" RCMEM="16050" RCPROC="1"
RCSWAP="32432" REQUESTID="1234" RESCOUNT="1" RMACCESSLIST="bdaw" RSVLIST="bdaw-TTL-
1234" SPEED="1.000000" STATACTIVETIME="2109" STATMODIFYTIME="1412181806"
STATTOTALTIME="2164684" STATUPTIME="2164668" TTL="1441778400" VARATTR="DEV"/>
  <node AVLCLASS="[DevQ][batch]" CFGCLASS="[DevQ][batch]" CPUCLOCK="OnDemand:800mhz"
FEATURES="DEV" LASTUPDATETIME="1412200545" MAXJOB="0" MAXJOBPERUSER="0"
MAXLOAD="0.000000" NODEID="nuc2" NODEINDEX="2" NODESTATE="Idle" OS="linux"
OSLIST="linux" PARTITION="bdaw" PRIORITY="0" PROCSPEED="0" RADISK="1" RAMEM="15193"
RAPROC="4" RASWAP="16717" RCDISK="1" RCMEM="15918" RCPROC="4" RCSWAP="17442"
RMACCESSLIST="bdaw" SPEED="1.000000" STATACTIVETIME="34" STATMODIFYTIME="1412114379"
STATTOTALTIME="86507" STATUPTIME="86475" VARATTR="DEV"/>
</Data>
```

16.6.2 mdiag -T

The *mdiag -T* command is used to display information about each trigger. See 3.7.18 mdiag -T for more information.

In the following example, the current list of triggers is displayed using the *mdiag -T* command. Notice the node end triggers associated with nodes.

```
$ mdiag -T
TrigID                 Object ID               Event  AType            ActionDate
State
---------------------- -------------------- --------- ------ -------------------- ------
-----
83                     node:DEFAULT              end   exec                      -
Blocked
85*                    node:nuc2                 end   exec            -1:00:01:10
Successful
84*                    node:bdaw                 end   exec               -5:17:23
Active
* indicates trigger has completed
```

16.6.3 checknode -v <node name>

The `checknode -v <node name>` command shows detailed state information and statistics including the `TTL`, the `access control list (ACL)` and the `requestid` for nodes that run jobs. See 3.7.2 checknode for more information.

In the following example, a reservation is created on the node at the `TTL` so that the jobs are not scheduled on the node beyond the `TTL`. Also, a node end trigger is configured on this node that will fire when the node is removed.

```
$ checknode -v bdaw
node bdaw

State:      Idle  (in current state for 5:18:38)
Configured Resources: PROCS: 1  MEM: 15G  SWAP: 31G  DISK: 1M
Utilized   Resources: MEM: 6230M  SWAP: 6230M
Dedicated  Resources: ---
Attributes:         DEV
ACL:        USER==bdaw+:==adaptive+
  MTBF(longterm): 1:00:31:02  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.340
Partition:  bdaw  Rack/Slot:  ---
Features:   DEV
IdleTime:   23:38:11
Classes:    [DevQ][batch]
RM[bdaw]*:  TYPE=PBS
EffNodeAccessPolicy: SHARED
RequestID:  1234
TTL:  Wed Sep  9 00:00:00 2024


Total Time: 25:01:24:09  Up: 25:01:23:53 (100.00%)  Active: 00:35:09 (0.10%)

Reservations:
  bdaw-TTL-1234x1  User     342days ->   INFINITY (  INFINITY)
    Blocked Resources@   342days  Procs: 1/1 (100.00%)  Mem: 16050/16050 (100.00%)
Swap: 32432/32432 (100.00%)  Disk: 1/1 (100.00%)
TrigID                Object ID              Event  AType           ActionDate
State
--------------------- -------------------- --------- ------- -------------------- ------
-----
84*                   node:bdaw                end   exec  Wed Oct  1 10:43:26
Active
  Launch Time:   -00:00:14
  Flags:         globaltrig
  Last Execution State: Active (ExitCode: 0)
  BlockUntil:    5:18:24  ActiveTime:  -1:00:29:57
  PID:           7088
  Action Data:   /home/bdaw/nodeend.sh $OID
  StdOut:        /opt/moab/spool/nodeend.sh.oMNnWkU
  StdErr:        /opt/moab/spool/nodeend.sh.ennUbAp

* indicates trigger has completed
```

16.7 Usage Policies

As part of your Elastic Computing solution, you can keep track of processor seconds on all dynamic nodes to limit over-bursting. For example, if your configuration allows 1000 processor seconds of use every day, then if a job needs to burst (and the used processor seconds reaches 1000 before the job can burst), the trigger to burst the job will not fire, and an error message is generated. You can view the error message using 'mdiag -T -v'.

In this section:

16.7.1 Available Policies

There are four different values you can set: day, month, quarter, or year. The second count resets at the beginning of each period. For ease of use, you can choose to set the limits based on processor hours, and the system will automatically convert the hours to seconds.

These are the available policies you can set in the moab.cfg file to limit over-bursting:

- To specify by processor seconds, use:
  - MAXDAILYELASTICPROCSECONDS
  - MAXMONTHLYELASTICPROCSECONDS
  - MAXQUARTERLYELASTICPROCSECONDS
  - MAXYEARLYELASTICPROCSECONDS
- To specify by processor hours, use:
  - MAXDAILYELASTICPROCHOURS
  - MAXMONTHLYELASTICPROCHOURS
  - MAXQUARTERLYELASTICPROCHOURS
  - MAXYEARLYELASTICPROCHOURS

16.7.2 Policy Levels

You can set the usage policies at the global partition or QoS level:

- Global Partition – Once the elastic node first appears, Moab will begin keeping track of its processor seconds or hours. If the processor seconds reaches the limit, it will not fire off the elastic trigger so no new nodes will come in. For example:

  ```
  PARCFG[ALL]                MAXDAILYELASTICPROCSECONDS=1000
  ```

  You can view the used and remaining limits using 'showstats -v'.
- QoS – Processor seconds or hours start being counted once a job is submitting using that particular QOS, not from when the node first appears. For example:

  ```
  QOSCFG[HIGH]               MAXDAILYELASTICPROCSECONDS=500
  ```

  A job is submitted requesting the "HIGH" QOS; the processor seconds begin ticking up for that QOS.

  You can view the used and remaining limits using 'mdiag -q -v'.

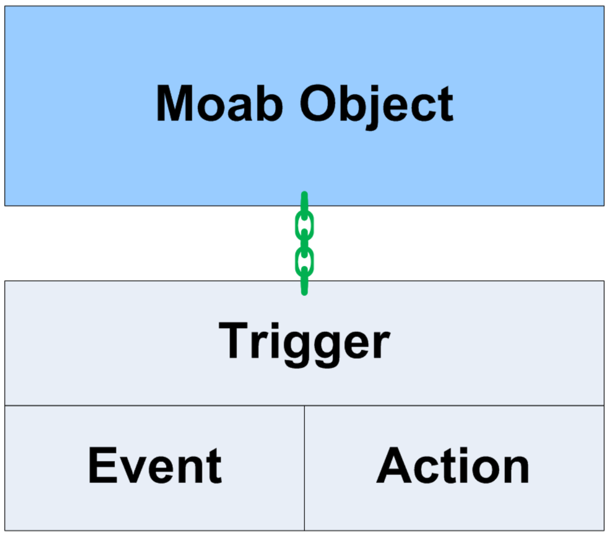
Chapter 17: Object Triggers

In this chapter:

17.1 About Object Triggers

Moab triggers are configurable actions that respond to an event occurring on a Moab object. A trigger is attached to an object and consists of both an event that can take place on the object and the action that the trigger will take.

Image 17-1: Trigger attachment



Triggers are a powerful tool. Extreme caution should be taken when using them. They are useful in creating automatic responses to well-understood Moab events; however, by default triggers run as root and do exactly as they are told, meaning they require great thought and consideration to ensure that they act appropriately in response to the event.

Use Case

An admin wants to create the following setup in Moab:

When a node's temperature exceeds 34°C, Moab reserves it. If the temperature increases to more than 40°C, Moab requeues all jobs on the node. If the node's temperature exceeds 50°C, Moab shuts it down. Moab removes the node's reservation and unsets the variables when the node cools to less than 25°C.

The admin wants to receive an email whenever any of these events occur. All of this can be configured in Moab using triggers. To see a full example for this use case, see 17.3.9 Node Maintenance Example.

17.2 Object Trigger Tasks

In this section:

17.2.1 Creating a Trigger

Three methods exist for attaching a trigger to an object:

- Directly to the object via the command line
- Directly to the object via the configuration file
- As part of a template via the configuration file

`<attr>=<val>` pair delimiters, quotation marks, and other elements of the syntax might differ slightly from one method/object combination to another, but creating any trigger follows the same basic format:

```
<attr>=<val>[[{&,}<attr>=<val>]...]
```

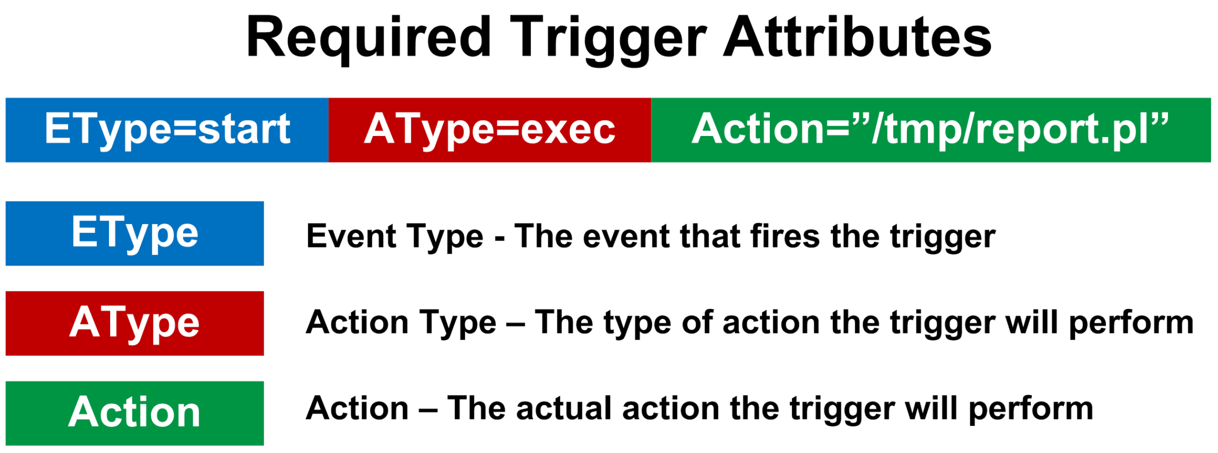
The beginning of the trigger is set off by the keyword *trigger*. It is followed by a delimited list (typically by commas) of `<attr>=<val>` pairs.

Each method of trigger creation can only be used for certain Moab objects. The following table displays which objects can receive triggers via each method:

Method	Objects
Command line	job, reservation; a trigger can be attached to any existing object using mschedctl -c
Configuration file	node, reservation, resource manager, scheduler
Template	job, reservation

Triggers are composed of attributes. Only three are required for each trigger: an EType (event type), an AType (action type), and an Action.

Image 17-2: Required trigger attributes


Required Trigger Attributes
EType=start
AType=exec
Action="/tmp/report.pl"
EType
Event Type - The event that fires the trigger
AType
Action Type – The type of action the trigger will perform
Action
Action – The actual action the trigger will perform

Other attributes exist to further customize triggers. See 17.3.7 Trigger Components for more information.

To Create a Moab Trigger

1. Choose an object to which, and a method by which, you will attach the trigger. Use the format and examples described in its corresponding documentation:

- Job Triggers
- Node Triggers
- Reservation Triggers
- Resource Manager Triggers
- Scheduler Triggers

> If the trigger is to be attached to a job, you must first enable job triggers (see 17.2.6 Enabling Job Triggers for more information). Carefully review the warning before doing so.

2. Decide whether to attach the trigger via the command line or configuration file. Verify the correct syntax.
3. Set the `EType` equal to whichever event will launch the trigger if and when it occurs on the object.

   Each object has a different lifecycle, so not every event type will occur on every object. For a list of valid `ETypes` for your selected object, see the corresponding object reference page linked in step 1.

   a. To modify the timing of the trigger in any of the following ways, see 17.3.7.B Event-Modifying Trigger Components:

      - To set the trigger as rearmable and specify the amount of time the trigger must wait before firing again.
      - To set an amount of time before or after the event that the trigger will fire (see the trigger attribute Offset for restrictions).
      - To set a specific threshold and the amount of time that the object must meet that threshold before the trigger will fire.
4. Configure the action that the trigger will take when the event happens. To do so, you must set the `AType` to a valid value for your object and specify the action. For instance, to execute a script, set the `AType` to `exec` and the `Action` to the location of the script in quotation marks. Include the name of the object on which the script will run.

```
NODECFG[node01] TRIGGER=EType=fail,AType=exec,Action="node.fail.sh node01"
```

   a. To modify the action in any of the following ways, see 17.3.7.C Action-Modifying Trigger Components:

- To specify environment variables available to the trigger
- To set a flag on the trigger:
  - To attach any stderr output generated by the trigger to the parent object
  - To destroy the trigger if its object ends or cancels
  - To tell Moab to checkpoint the trigger
  - To set the trigger as periodic
  - To pass the object's XML information to the trigger's stdin
  - To set the trigger to reset if its object is modified
  - To set the trigger to fire under the user ID of the object's owner
- To specify an amount of time that Moab will suspend normal operation to wait for the trigger to execute
- To allot an amount of time that the trigger will attempt to run before it is marked as unsuccessful and the process, if any exists, is killed
- Set a maximum number of times that a trigger will attempt to fire before it fails

b. To give the trigger a name or description, see 17.3.7.D Organizational Trigger Components.

c. To configure the trigger to set or unset a variable when it fires or to require a variable to fire, see 17.4.1.A Setting and Receiving Trigger Variables.

17.2.2 Using a Trigger to Send Email

Mail triggers can be attached to nodes, jobs, reservations, and the scheduler. The recipient of the email depends on the object to which the trigger is attached. To select different recipient(s) and add flexibility to formatting, send email via a script using an exec trigger.

To Use a Trigger to Send Email

1. For objects that send mail to the primary user, you must configure the Moab Administrator email using the MAILPROGRAM parameter.
2. Create a trigger on one of the four objects listed below, setting the `AType` to mail and the `Action` to the body of the message inside of quotation marks:

Object	Recipient
Node	The primary user (the first user listed in ADMINCFG[1], typically root)

Object	Recipient
Job	The job's owner
Reservation	The primary user
Scheduler	The primary user

When attaching a mail trigger to all objects of a certain type, use internal variables in the `Action` to add information that is specific to an object, such as the ID, owner, time the event occurred, etc. A variable must be preceded by a dollar sign ($).

Variable	Description
$OID	Name of the object to which the trigger is attached.
$OTYPE	The type of object to which the trigger is attached.
$TIME	Time the trigger launched.
$HOSTLIST	Hostlist of the trigger's object (jobs and reservations).
$OWNER	Owner of the trigger's object (jobs and reservations).
$USER	User (jobs and reservations).

The variable is replaced with the information described above. For example, the following trigger is configured on all nodes:

```
NODECFG[DEFAULT] EType=fail,AType=mail,Action="node $OID failed at $TIME"
```

When, for example, node `node03` fails, an email is sent to the primary user with a message with the subject line "node node03 started on Sat Aug 18 11:42:00".

17.2.3 Using a Trigger to Execute a Script

Exec triggers launch a program or script when the event occurs. A few examples of what a script might do in response to an event include:

- Execute an external program
- Send a complex email to any desired recipient(s)
- Collect diagnostics

It is important to note that when a script runs via a trigger, Moab forks and performs a direct OS exec, meaning there will be no pre-processing of the command by the shell. In addition, the script runs in a new, reduced environment without the same settings and variables as the environment from which it stemmed. The script must be able to run in the reduced environment.

To Use a Trigger to Execute a Script

1. Create or locate the script and note its location.
2. Create a trigger on the desired object, setting the `AType` to `exec` and the `Action` to location of the script or program:

   ```
   JOBCFG[temp1] TRIGGER=EType=start,AType=exec,Offset=03:00,Action="/tmp/monitor.pl"
   ```

   Jobs with the `temp1` template receive a trigger that executes `monitor.pl` three minutes after the job starts.

17.2.4 Using a Trigger to Perform Internal Moab Actions

To Perform Internal Actions in Moab with a Trigger

Create a trigger on a job, node, or reservation, setting the `AType` to `internal` and the `Action` to one of the following:

- `node:-:reserve` - reserves the node to which the trigger is attached
- `job:-:cancel` - cancels the job to which the trigger is attached
- `reservation:-:cancel` - cancels the reservation to which the trigger is attached

The specified object reserves or cancels itself once the event occurs. See the trigger component Internal Action for examples.

17.2.5 Requiring an Object Threshold for Trigger Execution

Threshold triggers allow sites to configure triggers to launch based on internal scheduler statistics, such as generic metrics. For example, you might configure a trigger to warn the admin when the percentage of nodes available is less than 25.

To Configure a Threshold Trigger

1. Create a trigger. Set its `EType` to `threshold`. Configure the `AType`, `Action`, and `Threshold` attributes' values based on the thresholds per object listed in the table in 17.3.6 Threshold Triggers.

   ```
   NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py
   $OID",Threshold=gmetric
   ```

2. Insert the gmetric name between brackets (such as `gmetric[temp]`). Provide a comparison operator. For options, see the Comparison Operator table.

3. Provide a number or string to match against the threshold:

   ```
   NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py
   $OID",Threshold=gmetric[TEMP]>70,RearmTime=5:00
   ```

   Moab launches a script that warns the admin when `node04`'s gmetric `temp` exceeds 70. Moab rearms the trigger five minutes after it fires.

17.2.6 Enabling Job Triggers

By default, common users cannot create most objects, and as a result, common users also cannot create triggers. The exception, however, is jobs. Because common users can create jobs and triggers generally run as root, additional security is necessary to ensure that not all users can create triggers. For this reason, job triggers are disabled by default.

⚠ Because triggers generally run as root, any user given the power to attach triggers has the power to run scripts and commands as root. We recommend that you only enable job triggers on closed systems where human users do not have access to directly submit jobs.

To give specific users permission to create job triggers, you must create a QoS, set the `trigger` flag, and add users to it.

To Enable Job Triggers

1. In the `moab.cfg` file, create a QoS and set the `trigger` flag:

   ```
   QOSCFG[triggerok]   QFLAGS=trigger
   ```

2. Add users to the QoS who will be allowed to add triggers to jobs:

   ```
   USERCFG[joe]        QDEF=triggerok
   ```

   User `joe` is added to the `triggerok` QoS, giving him both the power to create job triggers and root access to the machine.

17.2.7 Modifying a Trigger

You can modify a trigger at any time by updating its settings in the Moab configuration file (`moab.cfg`). This will update most triggers at the beginning of the next Moab iteration; however, modifying template triggers (configured using the parameter RSVPROFILE or JOBCFG) will not update the instances of the trigger that were attached to individual reservations or jobs on creation. The modification will only affect the triggers that the template attaches to future objects.

Any trigger with a specified name can be modified using the mschedctl -m command in the following format: `mschedctl -m trigger:<triggerID><attr1>=<val1><attr2>=<val2>`

Modifying triggers on the command line does not change their configuration in `moab.cfg`. Except for reservations that are checkpointed, changes made dynamically are lost when Moab restarts.

For example, the procedure below demonstrates how to modify the following trigger so that the offset is 10 minutes instead of 5 and so that Moab will attempt to fire the trigger up to 10 times if it fails. Assume your trigger currently looks like this:

```
NODECFG[DEFAULT] EType=fail,AType=exec,Action="/scripts/node_
fail.pl",Name=nodeFailTrig,Offset=00:05:00,MultiFire=TRUE,RearmTime=01:00:00
```

To Modify a Trigger

1. Type *`mschedctl -m`* into the command line and set off the trigger modification with `trigger:<id>`. Use the trigger's assigned ID or specified name to state which trigger will receive the modification.

   ```
   > mschedctl -m trigger:nodeFailTrig
   ```

2. Type any changing attributes equal to the new value. Separate multiple modifications with a space between each `<attr>=<val>` pair. In this case, set the `Offset` and `MaxRetry` attributes the following way:

   ```
   > mschedctl -m trigger:nodeFailTrig Offset=00:10:00 MaxRetry=10
   ```

   The newly-specified attributes replace the original ones. Trigger `nodeFailTrig` now has an offset of 10 minutes and will try to fire a maximum of 10 times if it fails. The new trigger has the following attributes:

   ```
   EType=fail,AType=exec,Action="/scripts/start_
   rsv.pl",Name=nodeFailTrig,Offset=00:10:00,MultiFire=TRUE,RearmTime=01:00:00,MaxRetr
   y=10
   ```

17.2.8 Viewing a Trigger

Moab provides a list of triggers when you run the mdiag -T command. You can view a specific trigger by running *mdiag -T* in the following format: `mdiag -T [<triggerID>|<objectID>|<triggerName>|<objectType>]`

To View a Trigger

1. Type `mdiag -T` in the command line.
2. Specify either the trigger ID, the trigger name, the name of the object to which the trigger is attached, or the type of object to which the trigger is attached. For example, if you wanted to view information about a trigger with ID `trigger.34` and name `jobFailTrigger`, which is attached to job `job.493`, you could run any of the following commands:

   ```
   > mdiag -T trigger.34

   > mdiag -T job.493

   > mdiag -T jobFailTrigger

   > mdiag -T job
   ```

   The output of the first command provides basic information about `trigger.34`; the second command, information about all triggers attached to `job.493` that the user can access; the third command, basic information about `jobFailTrigger`; and the fourth command, basic information about all triggers attached to jobs that the user can access.
3. (Optional) To view additional information about the trigger, run the same command with the `-v` flag specified after `-T`:

   ```
   > mdiag -T -v job.493
   ```

   This mode outputs information in multiple lines.
4. (Optional) To view detailed information about all triggers available to you, use the `mdiag -T -v` command. This outputs all triggers available to the user in a single line for each trigger. It provides additional state information about triggers, including reasons triggers are currently blocked.

   ```
   > mdiag -T -v
   ```

17.2.9 Checkpointing a Trigger

Checkpointing is the process of saving state information when Moab is shut down. In general, triggers defined in the `moab.cfg` file are not checkpointed but are recreated when Moab starts. The exception is the JOBCFG parameter, which attaches triggers to jobs as they are created.

There are two cases where you might want to tell Moab to checkpoint a trigger:

- If a trigger is defined in the `moab.cfg` file but was created at the command line
- When creating a trigger using the mschedctl command

To Checkpoint a Trigger

1. Locate the trigger to be checkpointed in the `moab.cfg` file, create one on the command line, or modify a trigger dynamically (see 17.2.7 Modifying a Trigger for more information). Attach the `checkpoint` flag using the `FLAGS` attribute. For more information about flags, see the trigger component Flags.

   ```
   FLAGS=checkpoint
   ```

2. If you are working in the configuration file, save the changes. Moab will now checkpoint your trigger.

17.3 Object Trigger Reference

In this section:

17.3.1 Job Triggers

For security reasons, job triggers are disabled by default. They must be enabled in order to successfully attach triggers to jobs (see 17.2.6 Enabling Job Triggers for more information).

Triggers attached to jobs follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Jobs, like other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details associated with attaching triggers to jobs.

In this topic:

17.3.1.A Creation Methods

Method	Format	Example
Command line on job creation: msub -l	`msub <jobName> -l 'trig=<trigSpec>'` Attributes are delimited by backslash ampersand (\&).	`> msub my.job -l 'trig=EType=create\&AType=exec\&Action="/jobs/my_trigger.pl"\&Offset=10:00'`
Command line on existing job: mschedctl -c	`mschedctl -c trigger <trigSpec> -o job:<jobID>`	`> mschedctl -c trigger EType=end,AType=mail,Action="Job moab.54 has ended" -o job:moab.54`
Job template in `moab.cfg` : JOBCFG	`JOBCFG [<templateName>] TRIGGER=<trigSpec>`	`JOBCFG[vmcreate] TRIGGER=,EType=end,AType=exec,Action="/tmp/jobEnd.sh"`
Class event: CLASSCFG	`CLASSCFG[<classID>] JOBTRIGGER=<trigSpec>`	`CLASSCFG[batch] JOBTRIGGER=atype=exec,etype=create,action="/opt/moab/tools/job_trigger.pl"`

17.3.1.B Valid Event Types

- cancel
- checkpoint
- create

- end
- fail
- hold
- modify
- preempt
- start

17.3.1.C Valid Action Types

- changeparam
- exec
- internal
- mail

17.3.1.D Mail Recipient

The job's owner. See 17.2.2 Using a Trigger to Send Email for more information.

17.3.2 Node Triggers

Triggers attached to nodes follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Nodes, like the other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to nodes.

In this topic:

17.3.2.A Creation Methods

Method	Format	Example
Command line on existing node: mschedctl - c	`mschedctl -c trigger <trigSpec> -o node:<nodeID>`	`> mschedctl -c trigger EType=fail,AType=exec,Action="/tmp/nodeFailure.sh" -o node:node01`
Node configuration in `moab.cfg`: NODECFG	`NODECFG [<name>] TRIGGER=<trigSpec>`	`NODECFG[node04] TRIGGER=EType=threshold,AType=exec,Action="$HOME/hightemp.py $OID",Threshold=gmetric[TEMP]>70`

17.3.2.B Valid Event Types

- create
- discover
- end
- fail
- standing
- threshold

17.3.2.C Valid Action Types

- changeparam
- exec
- internal
- mail

17.3.2.D Thresholds

Node Threshold Settings	
Valid ETypes	threshold

Node Threshold Settings	
Valid Threshold Types	gmetric

17.3.2.E Mail Recipient

The user listed first in ADMINCFG[1] (usually `root`). See 17.2.2 Using a Trigger to Send Email for more information.

17.3.3 Reservation Triggers

Triggers attached to reservations follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. Reservations, like the other objects, have a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to reservations.

In this topic:

17.3.3.A Creation Methods

Method	Format	Example
Command line on reservation creation: mrsvctl -T	`mrsvctl -c -h <hostlist> -T <trigSpec>`	`> mrsvctl -c -h node01 -T EType=start,AType=exec, Action="/scripts/node_start.pl"`
Command line on existing	`mschedctl -c trigger`	`> mschedctl -c trigger EType=modify,AType=mail,Action="Reservation system.4 has been modified" -o rsv:system.4`

Method	Format	Example
reservation: mschedctl - c	`<trigSpec> -o rsv:<rsvID>`	
Standing reservation configuration in `moab.cfg` : SRCFG	`SRCFG[<name>] TRIGGER=<trigSpec>`	`SRCFG[Mail2] TRIGGER=EType=start,Offset=200,AType=exec,Action="/tmp/email.sh"`
Reservation template in `moab.cfg` : RSVPROFILE	`RSVPROFILE[<name>] TRIGGER=<trigSpec>`	`RSVPROFILE[rsvtest] TRIGGER=EType=cancel,AType=exec,Action="$HOME/logdate.pl TEST CANCEL $VPCHOSTLIST $OID $HOSTLIST $ACTIVE"`

17.3.3.B Valid Event Types

- create
- end
- modify
- standing
- start
- threshold

17.3.3.C Valid Action Types

- cancel
- changeparam
- exec
- internal
- jobpreempt
- mail

17.3.3.D Thresholds

Node Threshold Settings	
Valid ETypes	threshold
Valid Threshold types	usage

17.3.3.E Mail Recipient

The owner of the reservation. If the owner is unknown or not a user, the first user listed first in ADMINCFG (usually `root`). See 17.2.2 Using a Trigger to Send Email for more information.

17.3.4 Resource Manager Triggers

Triggers attached to the resource manager follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. The resource manager, like other objects, has a unique set of trigger rules. The table below details the methods, options, and other notable details that come with attaching triggers to resource managers.

In this topic:

17.3.4.A Creation Methods

Method	Format	Example
Command line on existing resource manager: mschedctl - c	`mschedctl -c trigger <trigSpec> -o rm:<rmID>`	`> mschedctl -c trigger EType=start,AType=exec,Action="/tmp/rmStart.sh" -o rm:torque`

Method	Format	Example
Resource manager configuration in `moab.cfg`: RMCFG	`RMCFG [<name>] TRIGGER= <trigSpec>`	`RMCFG[base] TRIGGER=EType=fail,AType=exec,Action="/opt/moab/tools/diagnose_rm.pl $OID"`

17.3.4.B Valid Event Types

- fail
- threshold

17.3.4.C Valid Action Types

- changeparam
- exec
- internal

17.3.5 Scheduler Triggers

Triggers attached to the scheduler follow the same basic rules and formats as attaching them to other objects; however, not all attribute options are valid for each object. The scheduler, like the other objects, has a unique set of trigger rules. The table below details the methods, options, and other notable details associated with attaching triggers to the scheduler.

In this topic:

17.3.5.A Creation Methods

Method	Format	Example
Command line on existing scheduler: mschedctl - c	`mschedctl -c trigger <trigSpec> -o sched:<schedID>`	`> mschedctl -c trigger EType=end,AType=exec,Action="/tmp/startRsvs.sh" -o sched:moab`
Scheduler configuration in `moab.cfg` : SCHEDCFG	`SCHEDCFG[<name>] TRIGGER=<trigSpec>`	`SCHEDCFG[MyCluster] TRIGGER=EType=fail,AType=mail,Action="scheduler failure detected on $TIME",RearmTime=15:00`

17.3.5.B Valid Event Types

- create
- end
- fail
- modify
- standing
- start

17.3.5.C Valid Action Types

- changeparam
- exec
- internal
- mail

17.3.5.D Mail Recipient

The user listed first in ADMINCFG (usually `root`). See 17.2.2 Using a Trigger to Send Email for more information.

17.3.6 Threshold Triggers

The following table identifies the object event, and usage types with which the threshold event/action type feature works:

Object Type	Event Type	Usage Types
Node	Threshold	gmetric
Reservation	Threshold	usage

The following table defines each of the usage types:

Usage Type	Description
gmetric	Generic performance metrics configured in Moab (see 10.8 Enabling Generic Metrics for more information).
usage	The percentage of the resource being used (not idle).

The following table defines each of the threshold trigger comparison operators:

Comparison Operator	Value
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to

Examples

Example 17-1: Reservation usage threshold

```
SRCFG[res1] TRIGGER=EType=threshold,AType=mail,Action="More than 75% of reservation
res1 is being used",Threshold=usage>75,FailOffset=1:00
```

When more than 75% of the reservation has been in use for at least a minute, Moab fires a trigger to notify the primary user.

17.3.7 Trigger Components

In this topic:

17.3.7.A Required Trigger Components

AType

Action Type	Description
cancel	Cancels the object.
changeparam	Causes Moab to give a parameter to a new value.
exec	Launches an external program or script on the command line when the dependencies are fulfilled. See 17.2.3 Using a Trigger to Execute a Script for more information.
internal	Modifies Moab without using the command line. See 17.2.4 Using a Trigger to Perform Internal Moab Actions for more information.
jobpreempt	Indicates the preempt policy to apply to all jobs currently allocated resources assigned to the trigger's parent reservation.
mail	Causes Moab to send mail. See 17.2.2 Using a Trigger to Send Email for more information.

Action

Cancel Action	
Format	NONE
Description	Indicates that Moab should cancel the reservation when the event occurs. No action should be specified.

Cancel Action	
Example	`Etype=threshold,Threshold=usage<10,FailOffset=1:00,AType=cancel` *When less than 10% of the reservation has been in use for a minute, Moab cancels it.*

Changeparam Action	
Format	`Action="<STRING>"`
Description	The parameter to change and its new value (using the same syntax and behavior as the changeparam command).
Example	`Atype=changeparam,Action="JOBCPURGETIME 02:00:00"` *Moab maintains detailed job information for two hours after a job has completed.*

Jobpreempt Action	
Format	`Action="cancel|checkpoint|requeue|suspend"`
Description	Signifies the parameter PREEMPTPOLICY to apply to jobs that are running on allocated resources.
Example	`RSVPROFILE[adm1] TRIGGER=EType=start,Offset=-240,AType=jobpreempt,Action="cancel"` *40 minutes after the reservation `adm1` starts, all jobs using the reservation's resources adopt a `PREEMPTPOLICY` of `cancel`.*

Mail Action	
Format	`Action="<MESSAGE>"`
Description	When `AType=mail`, the `Action` parameter contains the message body of the email. This can be configured to include certain variables. See 17.2.2 Using a Trigger to Send Email for details. Mail triggers can be configured to launch for node failures, reservation creation or release, scheduler failures, and even job events. In this way, site admins can keep track of scheduler events through email. The email comes from *moabadmin*, has a subject of *moab update*, and has a

Mail Action	
	body of whatever you specified in the `Action` attribute. The recipient list depends on the type of object the trigger is attached to. • **Node** - The primary user (first listed in ADMINCFG[1]), typically `root` • **Scheduler** - The primary user • **Job** - The user who owns the job • **Reservation** - The primary user
Example	`NODECFG[DEFAULT] TRIGGER=EType=fail,AType=mail,Action="node $OID will failed .",Offset=05:00:00` *This example sends an email to the primary admin informing him/her that the node (including the node ID) has failed.*

Exec Action	
Format	`Action="<script>"`
Description	Exec triggers will launch an external program or script when their dependencies are fulfilled. The following example will submit `job.cmd` and then execute `monitor.pl` three minutes after the job is started. See 17.2.3 Using a Trigger to Execute a Script for more information.
Example	`> msub -l trig=EType=start\&AType=exec\&Action="/tmp/monitor.pl" job.cmd\&Offset=03:00`

Internal Action	
Format	`Action="<objectType>:-:<cancel|reserve>"`
Description	A couple different actions are valid depending on what type of object the internal trigger is acting upon. The following list shows the available actions: • Reserve a node • Cancel a job • Cancel a reservation See 17.2.4 Using a Trigger to Perform Internal Moab Actions for more information.
Example	`NODECFG[node01] TRIGGER=EType=start,AType=internal,Action="node:-:reserve"`

Internal Action	

	When `node01` starts, it becomes a reservation.
	`> msub moab.3 -l 'trig=EType=fail\&AType=internal\&Action="job:-:cancel"`
	If `moab.3` fails, Moab cancels it.
	`> mrsvctl -c -a user==joe -h node50 -T EType=start,AType=internal,Action=" reservation:-:cancel",Offset=10:00`
	User `joe`'s jobs are given a ten-minute window to start, then the reservation cancels.

EType

Event Type	Description
cancel	The event is triggered when the parent object is either canceled or deleted.
checkpoint	Triggers fire when the job is checkpointed. `checkpoint` triggers can only be attached to jobs.
create	Triggers fire when the parent object is created. `create` triggers can be attached to nodes, jobs, reservations, classes, and the scheduler (when attached to the scheduler, triggers fire when Moab starts).
discover	Triggers fire when the node is loaded from a resource manager and Moab cannot recognize it nor find it in the checkpoint file.
end	Triggers fire when the parent object ends. `end` triggers can be attached to nodes, jobs, reservations, and the scheduler. When attached to the scheduler, triggers fire when Moab shuts down. When attached to jobs, triggers fire when jobs return successfully (completion code of 0).
fail	`fail` triggers can be attached to jobs, nodes, resource managers, and the scheduler. Triggers fire when the resource manager is in a corrupt or down state for longer than the configured fail time, when Moab detects a corruption in a node's reservation table, or when the job returns an unsuccessful completion code.
hold	Triggers fire when the job is put on hold. `hold` triggers can only be attached to jobs.

Event Type	Description
modify	Triggers fire when the parent object is modified. `modify` triggers can be attached to jobs and reservations.
preempt	Triggers fire when the job is preempted. `preempt` triggers can only be attached to jobs.
standing	Triggers fire multiple times based on a certain period. They can be used with Period and Offset attributes. `standing` triggers can be attached to nodes and the scheduler.
start	Triggers fire when the parent object or Moab starts. `start` triggers can be attached to jobs, reservations, resource managers, and the scheduler (when Moab starts and at the beginning of Moab's first iteration).
threshold	Triggers fire when a threshold, such as usage or a gmetric comparison, is true. `threshold` triggers can be attached to nodes and reservations. Triggers with `EType`s set to `threshold` must include the Threshold attribute.

TType

Trigger Type	Description
elastic	Specifies trigger for Elastic Computing. Only applicable for QoS triggers.
generic	Default trigger type. Not applicable for Elastic Computing.

17.3.7.B Event-Modifying Trigger Components

The following trigger attributes modify the event that causes the trigger to fire:

RearmTime	
Possible Values	`[[HH:]MM:]SS`
Description	The amount of time that must pass before a trigger can fire again. `RearmTime` is enforced from the trigger event time.
Usage Notes	---

Offset	
Possible Values	`[-][[HH:]MM:]SS`
Description	The relative time offset from event when trigger can fire.
Usage Notes	• Only end triggers can have a negative value for `Offset`. • `Offset` cannot be used with cancel.

Period	
Possible Values	`Minute`, `Hour`, `Day`, `Week`, `Month`, `Infinity`
Description	The period at which the trigger will regularly fire.
Usage Notes	---

Threshold	
Possible Values	`Threshold={<metric>[<metricName>]}{> >= < <= ==}<FLOAT>` Where `<metric>` is one of: • gmetric • usage
Description	When the object meets, drops below, or increases past the configured `Threshold`, the trigger will fire.
Usage Notes	Threshold triggers allow sites to configure triggers to launch based on internal scheduler statistics, such as the usage of a reservation.

FailOffset	
Possible Values	`[[HH:]MM:]SS`
Description	The time that the threshold condition must exist before the trigger fires.
Usage Notes	Use with fail triggers to avoid transient triggers.

17.3.7.C Action-Modifying Trigger Components

Flags	
Possible Values	`Flags=<flag>[:<flag>]` or `Flags=[<flag>][[<flag>]]` `attacherror` - If the trigger outputs anything to stderr, Moab attaches it as a message to the trigger object. `cleanup` - If the trigger is still running when the parent object completes or is canceled, Moab kills the trigger. `checkpoint` - Moab always checkpoints this trigger. For more information, see 17.2.9 Checkpointing a Trigger. `objectxmlstdin` - Trigger passes its parent's object XML information into the trigger's stdin. This only works for exec triggers with reservation type parents. `removestdfiles` - When the trigger is deleted, Moab will remove the stdin, stdout, and stderr files used by the trigger. `resetonmodify` - The trigger resets if its object is modified, even if `RearmTime` is not set. `user` - The trigger executes under the user ID of the object's owner. If the parent object is the scheduler, you can explicitly specify the user using the format `user+<username>`. For example: `Flags=user+john`.
Description	Specifies various trigger behaviors and actions.
Usage Notes	When specifying multiple flags, each flag can be delimited by colons (:) or with square brackets; for example: `Flags=[user][cleanup]` or `Flags=user:cleanup`

BlockTime	
Possible Values	`[[HH:]MM:]SS`
Description	The amount of time Moab will suspend normal operation to wait for trigger execution to finish.
Usage Notes	Use caution; Moab will completely stop normal operation until `BlockTime` expires.

ExpireTime	
Possible Values	`<INTEGER>`

ExpireTime	
Description	The time at which trigger should be terminated if it has not already been activated.
Usage Notes	---

Timeout	
Possible Values	`[+|-][[HH:]MM:]SS`
Description	The time allotted to this trigger before it is marked as unsuccessful and its process (if any) killed.
Usage Notes	---

MaxRetry	
Possible Values	`MaxRetry=<INTEGER>`
Description	The number of times `Action` will be attempted before the trigger is designated a failure.
Usage Notes	If `Action` fails, the trigger will restart immediately (up to `MaxRetry` times). If it fails more than `MaxRetry` times, the trigger has failed. This restart ignores `FailOffset` and `RearmTime`.

17.3.7.D Organizational Trigger Components

Name	
Possible Values	`Name=<STRING>`
Description	Name of the trigger.
Usage Notes	Because Moab uses its own internal ID to distinguish triggers, the `Name` need not be unique. Only the first 16 characters of `Name` are stored by Moab.

Description	
Possible Values	`Description=<STRING>`
Description	Description of the trigger.
Usage Notes	---

17.3.8 Trigger Exit Codes

By default, Moab considers any non-zero exit code as a failure and marks the trigger as having failed. If a trigger is killed by a signal outside of Moab, Moab treats the signal as the exit code and (in almost all cases) marks the trigger as having failed. Only exec triggers that exit with an exit code of 0 are marked as successful.

17.3.9 Node Maintenance Example

Example Scenario

An admin wants to create the following setup in Moab:

When a node's temperature exceeds 34°C, Moab reserves it. If the temperature increases to more than 40°C, Moab requeues all jobs on the node. If the node's temperature exceeds 50°C, Moab shuts it down. Moab removes the node's reservation and unsets the variables when the node cools to less than 25°C. The admin wants to receive an email whenever any of these events occur.

The first trigger reserves the node when its reported temperature exceeds 34°C. Note that the gmetric name in the trigger must match the name of the configured gmetric exactly, including its case (see 10.8 Enabling Generic Metrics for more information).

```
NODECFG[DEFAULT] TRIGGER=Description="ThresholdA",EType=threshold,Threshold=gmetric
[temp]>34,AType=internal,Action="node:-:reserve",RearmTime=30,Offset=2:00,Sets=temp_
rsv
```

The admin wants the trigger to fire any time a node overheats, so it must be rearmable. It also needs to specify that the node must be over 34°C for at least two minutes for Moab to reserve it. If the trigger succeeds, it will set a variable to be received by the next trigger in order to make them sequential.

The admin wants to know when this trigger has fired, so another trigger will send an email once the first trigger has fired and the `temp_rsv` variable is set. This one does so via a script:

```
NODECFG[DEFAULT] Trigger=Description="Email on
Reservation",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailReserve.pl
$OID",RearmTime=3:00,Requires=temp_rsv
```

The second threshold trigger requeues the node's jobs if the node exceeds 40°C and the `temp_rsv` variable is set. It uses a script to do so. It sets `node_evac` variable when it fires, regardless of whether it succeeds or fails.

```
NODECFG[DEFAULT] Trigger=Description="Threshold B",EType=threshold,Threshold=gmetric
[temp]>40,Atype=exec,Action="$TOOLSDIR/node_evacuate.pl
$OID",RearmTime=3:00,requires=temp_rsv,Sets=node_evac,!node_evac
```

The admin wants another email to inform him that the node is still overheating and has been evacuated. Another email trigger fires once it receives the `node_evac` variable.

```
NODECFG[DEFAULT] Trigger=Description="Email on
Evacuation",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailEvac.pl
$OID",RearmTime=3:00,Requires=node_evac
```

The third threshold trigger uses a script to shut down the node if the temp gmetric exceeds 50 and the `node_evac` variable is set. It sets a `node_shutdown` variable to be received by the notification email.

```
NODECFG[DEFAULT TRIGGER=Description="Threshold C",EType=threshold,Threshold=gmetric
[temp]>50,AType=exec,Action="$TOOLSDIR/node_shutdown.pl
$OID",RearmTime=3:00,Requires=node_evac,Sets=node_shutdown

NODECFG[DEFAULT] Trigger=Description="Email on
Shutdown",EType=start,AType=exec,Action="$TOOLSDIR/node_temp_emailShutdown.pl
$OID",RearmTime=3:00,Requires=node_shutdown
```

The final trigger removes the reservation and unsets the variables once the node's `temp` gmetric is less than 25:

```
NODECFG[DEFAULT] Trigger=Description="Remove
Reservation",EType=threshold,Threshold=gmetric
[temp]<25,AType=exec,Action="opt/moab/bin/mrsvctl -r
r:$OID",RearmTime=3:00,Requires=temp_rsv,unsets=temp_rsv.node_evac.node_shutdown
```

17.3.10 Environment Creation Example

Example Scenario

An admin wants to create the following setup in Moab:

If a user requests an environment, she must have the permission of her two managers and the admin. If all three approve, then the environment builds. The user is sent email messages informing her of the environment's end date in case she wants an extension. These are sent 7, 3, and 1 days prior to the environment's ending.

The admin wants to require his and the managers' approval of any modifications the user makes to her environment so that it cannot be extended without consent.

The first trigger requests manager and admin approval in response to the user's environment request. So in the event of a reservation's creation, a script is used to send messages to the admin and manager. The internal variable `OWNER` is used to indicate to the recipients (via the script) which user is requesting the environment.

```
RSVPROFILE[envSetup] TRIGGER=EType=create,AType=exec,Action="envRequest.sh $OWNER"
```

The managers and admin use an external program to approve or reject the request. On approval, a variable is sent back to Moab (to the reservation specifically). Once all three variables are set, the environment can start. In this example, the variables are called `approval1`, `approval2`, and `approval3`.

```
RSVPROFILE[envSetup]
TRIGGER=EType=start,AType=exec,Action="buildScript",Requires=approval1.approval2.appro
val3
```

As it is configured now, the reservation will continue to reserve the requested resources regardless of whether all three approvals are given. So, in case approval is not given, the next trigger cancels the reservation 7 days after its creation if the three variables are not set.

```
RSVPROFILE[envSetup]
TRIGGER=EType=create,Offset=7:00:00,AType=internal,Action="rsv:-:cancel",Requires=!app
roval1.!approval2.!approval3
```

Every remaining trigger in this series is meant to fire for an approved environment and must require the approval variables. Otherwise these notifications would be sent to users who do not have the environment they requested. The next triggers must be rearmable so that it can fire again if necessary; however, they should be set to just over the amount of time left on the reservation so that it doesn't fire again for the same environment. The notification triggers use the `Offset` attribute to fire at the admin's requested times (7, 3, and 1 day(s) prior to the environment's end).

```
RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
```

```
7:00:00,AType=exec,Action="weekNotification.sh",RearmTime=7:00:00:02,Requires=approval
1.approval2.approval3

RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
3:00:00,AType=exec,Action="3dayNotification.sh",RearmTime=3:00:00:02,Requires=approval
1.approval2.approval3

RSVPROFILE[envSetup] TRIGGER=EType=end,Offset=-
1:00:00,AType=exec,Action="dayNotification.sh",RearmTime=1:00:00:02,Requires=approval1
.approval2.approval3
```

The next trigger requests admin and manager approval when the environment is modified. The problem is that the trigger must be rearmable in case of multiple modifications and each time the RearmTime is reached, Moab will fire the trigger based on the *first* instance of modification. To resolve this issue, this modification trigger requires a `modify` variable. When the reservation is modified, the `modify` variable is set.

```
RSVPROFILE[envSetup]
TRIGGER=EType=modify,AType=exec,Action="modify.sh",RearmTime=1:00:00,Requires=approval
1.approval2.approval3.!modify,Sets=modify
RSVPROFILE[envSetup]
TRIGGER=EType=modify,AType=exec,Action="modificationRequest.sh",RearmTime=5:00,Require
s=approval1.approval2.approval3.modify,Unsets=modify
```

The final triggers notify the user of the end of the environment:

```
RSVPROFILE[envSetup]
TRIGGER=EType=end,AType=exec,Action="end.sh",Requires=approval1.approval2.approval3
```

The same trigger is repeated for the `cancelEType` in case the environment ends unexpectedly:

```
RSVPROFILE[envSetup]
TRIGGER=EType=cancel,AType=exec,Action="end.sh",Requires=approval1.approval2.approval3
```

17.4 Trigger Variables

Trigger variables are pieces of information that pass from trigger to trigger. They allow triggers to fire based on another trigger's behavior, state, and/or output. A variable can be a required condition for a trigger to fire; for instance, a trigger might be set to launch when a reservation starts, but only if it has received a variable from another trigger indicating that a specific node has started first. Variables give greater flexibility and power to a site admin who wants to automate certain tasks and system behaviors.

Variables can be used to define under what circumstances the trigger will fire. Many Moab objects have their own variables and each object's variable name space is unique. Triggers can use their own variables or the variables attached to their parent objects. A trigger's variable name space is limited to itself and its parent object. Variables do not have to be unique across all objects.

In this section:

17.4.1 Trigger Variable Tasks

In this topic:

17.4.1.A Setting and Receiving Trigger Variables

Following is an example of how comparative dependencies can be expressed when creating a trigger.

To Set and Require Variables

1. Create a trigger:

   ```
   EType=start,AType=exec,Action="/tmp/trigger1.sh"
   ```

2. Use the `Sets` attribute to set a variable if the trigger succeeds. You can precede the variable with "!" to indicate that the variable should be set if the trigger fails. You can specify more than one variable by separating them with a period.

   ```
   AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=!Var1.Var2
   ```

   The trigger sets variable `Var2` when it succeeds and variable `Var1` when it fails.

3. Set up the recipient trigger(s). Use the Requires attribute to receive the variable(s). Note that preceding the variable with "!" means that the variable must not be set in order for the trigger to fire.

   ```
   AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=!Var1.Var2
   AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1
   ```

```
AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2
```

The second trigger will launch if `Var1` has been set (the first trigger failed), and the third trigger will launch if `Var2` is set (the first trigger succeeded).

4. Refine the requirement with comparisons.
    a. Use the following format: `<varID>[:<type>[:<varVal>]]`.
    b. Change `<varID>` to the variable name.
    c. Use any of the comparisons found on the page in place of `<type>`.
    d. Set the value that the variable will be compared against:

    ```
    AType=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1:eq:45
    AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2:ne:failure1
    ```

    The first trigger fires if `Var1` exists and has a value of 45. The second trigger fires if `Var2` does not have a string value of `failure1`.

17.4.1.B Externally Injecting Variables Into Job Triggers

Job triggers are able to see the variables in the job object to which it is attached. This means that, for triggers that are attached to job objects, another method for supplying variables exists. Updating the job object's variables effectively updates the variable for the trigger.

To Externally Inject Variables into Job Triggers

Use the mjobctl -m command to set a variable to attach to a job:

```
> mjobctl -m var=Flag1=TRUE 1664
```

The variable `Flag1` is set. This will be available to any trigger attached to job `1664`.

17.4.1.C Exporting Variables to Parent Objects

To Export Variables to Parent Objects

1. When setting a variable, indicate that the variable is to be exported to the parent object by using a caret (^):

    ```
    AType=exec,Action="/tmp/trigger1.sh",EType=start,Sets=Var1.!^Var2
    Atype=exec,Action="/tmp/trigger2.sh",EType=start,Requires=Var1
    AType=exec,Action="/tmp/trigger3.sh",EType=start,Requires=Var2
    ```

    `Var2` is exported to the parent object if the trigger fails. It can be used by job and reservation triggers at the same level or by parent objects.

2. (Optional) If running a script, you can set a variable as a string to pass up to the parent object.

   a. Set the variable to pass up to the parent object with the caret (^). Use the `execAType` to run a script.

   ```
   AType=exec,Action="/tmp/trigger.sh",EType=start,Sets=^Var1
   ```

   The trigger sets `Var1` when it completes successfully. Because the trigger launches a script, a string value can be set for `Var1`.

   b. Declare the variable's string value on its own line in the trigger stdout:

   ```
   EXITCODE=15
   Var1=linux
   ```

   `Var1` has the value of `linux` and is passed up to the parent object. This is useful in workflows where a trigger might depend on the value given by a previous trigger.

   To return multiple variables, simply print out one per line.

17.4.1.D Requiring Variables from Generations of Parent Objects

By default, triggers look for variables to fulfill dependencies in the object to which they are directly attached. If they are attached to a job object, they will also look in the job group, if defined. However, it is not uncommon for objects to have multiple generations of parent objects. If the desired behavior is to search through all parent objects, do the following task.

To Require Variables from Generations of Parent Objects

Set the `Requires` attribute in the trigger to the required variable, preceded by a caret (^):

```
EType=start,AType=exec,Action="/tmp/trigger2.sh",Requires=^Var1
```

The trigger searches through the parent objects in which it resides for the variable `Var1`.

17.4.1.E Requesting Name Space Variables

To Request a Name Space Variable in a Trigger

1. Configure the trigger. If it is attached to a generic system job, verify that it meets all Generic System Job Trigger Requirements below.
2. Create an argument list in the `Action` attribute (after the script path and before the closing quotes) and request the desired variable with an asterisk (`*`) in place of the

name space:

```
...Action="$HOME/myTrig.py $*.IPAddr"...
```

Each applicable name space variable is added to the argument list in the format *<varName>=<val>*.

For instance, the example above causes the script to run the following way:

```
> myTrig.py vc1.IPAddr=/tmp/dir1 vc2.IPAddr=/tmp/dir2 vc4.IPAddr=/tmp/dir3
```

Any other arguments provided here without name spaces will not change.

3. Filter which name spaces are passed down to a job trigger by setting `trigns` when you submit the job. Its value is a comma-delimited list of the desired name spaces.

```
msub -l ... -W x="trigns=vc2,vc4"
```

If the new job is applied to the example in step 2, the script's arguments include `vc2.IPAddr` and `vc4.Addr` and exclude `vc1.IPAddr`. The script runs as follows:

```
> myTrig.py vc2.IPAddr=/tmp/dir1 vc4.IPAddr=/tmp/dir2
```

17.4.1.F Generic System Job Trigger Requirements

A generic system job specifies one trigger that must meet all of the following criteria:

1. The `EType` is start.
2. The `AType` is exec.
3. The `Timeout` attribute is the desired walltime of the job. Moab ignores walltime requests when you submit a generic system job, using the trigger Timeout instead.

The trigger fires when the system job begins, and, because the trigger's Timeout doubles as the job's walltime, both complete at the same time. The job and trigger have the same completion code.

```
JOBCFG[gen] GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/installVM.py
$HOSTLIST",Timeout=1:00:00,Flags=objectxmlstdin
```

The job template `gen` creates a job with a walltime of 1 hour.

Sometimes the trigger will set a variable on completion or require a variable to run at all. For information about setting variables, passing them up to parent objects, and requiring variables on parent objects, see 17.4 Trigger Variables.

You can attach additional triggers using the `TRIGGER` attribute and delimit them with semicolons:

```
JOBCFG[gen] GENERICSYSJOB=<genericSystemJobTriggerSpecs>
```

```
JOBCFG[gen] TRIGGER=<triggerSpecs>;TRIGGER=<triggerSpecs>;TRIGGER=<triggerSpecs>
```

17.4.2 Trigger Variable Reference

In this topic:

17.4.2.A Dependency Trigger Components

Sets	
Possible Values	'.' delimited string
Description	Variable values this trigger sets upon success or failure.
Usage Notes	Preceding the string with an exclamation mark (!) indicates this variable is set upon trigger failure. Preceding the string with a caret (^) indicates this variable is to be exported to the parent object when the trigger completes and satisfies all its set conditions. Used in conjunction with Requires to create trigger dependencies.

Unsets	
Possible Values	'.' delimited string
Description	Variable this trigger destroys upon success or failure.
Usage Notes	Preceding the string with an exclamation mark (!) indicates this variable is unset upon trigger failure. Used in conjunction with Requires to create trigger dependencies.

Requires	
Possible Values	'.' delimited string
Description	Variables this trigger requires to be set or not set before it will fire.

Requires	
Usage Notes	Preceding the string with an exclamation mark (!) indicates this variable must not be set. Preceding the string with a caret (^) indicates that the variable can come from a parent object (see 17.4.1.D Requiring Variables from Generations of Parent Objects for more information). Used in conjunction with Sets to create trigger dependencies.

17.4.2.B Trigger Variable Comparison Types

The following table describes the types of comparisons you can use to express the relationship of a trigger variable to its value:

Type	Comparison	Notes
set	is set (exists)	Default
notset	not set (does not exist)	Same as specifying '!' before a variable
eq	equals	
ne	not equal	
gt	greater than	Integer values only
lt	less than	Integer values only
ge	greater than or equal to	Integer values only
le	less than or equal to	Integer values only

17.4.2.C Internal Variables

Internal Trigger Variables

Several internal variables are available for use in trigger scripts. These can be accessed using `$<varName>`.

Internal Variables	
COMPLETION CODE	When used in conjunction with the config parameter 'JOBCFG' and a trigger event type of 'end', it provides the trigger with the return code of the job.

Internal Variables	
ETYPE	The type of event that signals that the trigger can fire. ETYPE values include cancel, checkpoint, create, end, fail, hold, migrate, preempt, standing, start, and threshold.
OID	The name of the object to which the trigger was attached.
OTYPE	The type of object to which the trigger is attached; can be rsv, job, node, or sched.
OWNERMAIL	A variable that is populated only if the trigger's parent object has a user associated with it and that user has an email address associated with it.
REQUESTGEOMETRY	Passed to the Elastic Computing trigger script as a variable. Action="$HOME/geometry.pl $REQUESTGEOMETRY Example: `QOSCFG[sample] TRIGGER=EType=start,AType=exec,TType=elastic,Action="$HOME/geometry.pl $REQUESTGEOMETRY",timeout=5:00`
TIME	The time of the trigger launch in the following format: `Wed Mar 10 12:35:12 2025`
USER	The user (when applicable).

Object-Specific Internal Variables

Job Variables	
HOSTLIST	The entire hostlist of the job.
JOBID	String that identifies the job. You must also have `REQUESTGEOMETRY=PRIORITYJOBSIZE` to get the `JOBID`.
MASTERHOST	The primary node for the job.

Reservation Variables	
HOSTLIST	The entire hostlist for the reservation.

Reservation Variables	
OBJECTXML	The XML representation of an object output is the same that is generated by mdiag -r --xml
OS	The operating system on the first node of the reservation.
OWNER	The owner of the reservation.

Example 17-2: Internal variable

```
AType=exec,Action="/tmp/trigger.sh $OID $HOSTLIST",EType=start
```

The object ID (`$OID`) and hostlist (`$HOSTLIST`) will be passed to `/tmp/trigger.sh` as command line arguments when the trigger executes the script. The script can then process this information as needed.

Chapter 18: Miscellaneous

In this chapter:

18.1 User Feedback Facility

The Feedback facility enables a site admin to provide job performance information to users at job completion time. When a job completes, the program pointed to by the FEEDBACKPROGRAM parameter is called with a number of command line arguments. The site admin is responsible for creating a program capable of processing and acting upon the contents of the command line. The command line arguments passed are as follows:

1. job id
2. user name
3. user email
4. final job state
5. QoS requested
6. epoch time job was submitted
7. epoch time job started
8. epoch time job completed
9. job XFactor
10. job wallclock limit
11. processors requested
12. memory requested
13. average per task cpu load
14. maximum per task cpu load

15. average per task memory usage
16. maximum per task memory usage
17. messages associated with the job (if none, `[NONE]`)
18. hostlist (comma-delimited)
19. gres requests (`<GRES>:<COUNT>[,<GRES>:<COUNT>...]`)

For many sites, the feedback script is useful as a means of letting users know the accuracy of their wallclock limit estimate, and the CPU efficiency, and memory usage pattern of their job. The feedback script can be used as a mechanism to do any of the following:

- email users regarding statistics of all completed jobs
- email users only when certain criteria are met (such as "Job 14991 has just completed, which requested 128 MB of memory per task. During execution, it used 253 MB of memory per task potentially conflicting with other jobs. Please improve your resource usage estimates in future jobs.")
- update system databases
- take system actions based on job completion statistics

Some of these fields may be set to zero if the underlying OS/resource manager does not support the necessary data collection.

Example 18-1: FEEDBACKPROGRAM

```
FEEDBACKPROGRAM  /opt/moab/tools/fb.pl
```

18.2 Enabling High Availability Features

This section provides information and instructions for Adaptive Computing's HA solution with failover. If you want to use Linux HA or any other software stack, contact your account manager.

In this section:

18.2.1 High Availability Overview

High availability enables Moab to run on two different machines: a primary and secondary server. The configuration method to achieve this behavior takes advantage of a networked file system to configure two Moab servers with only one operating at a time.

> If you use a shared file system for high availability and Moab is configured to use a database, Moab must be an ODBC build, not SQLite.

> We recommend that you define LOGDIR to be a directory that exists on each server, but isn't a part of the NFS share.

When configured to run on a networked file system — any networked file system that supports file locking is supported — the first Moab server that starts locks a particular file. The second Moab server waits on that lock and only begins scheduling when it gains control of the lock on the file. This method achieves near instantaneous turnover between failures and eliminates the need for two Moab servers to synchronize information periodically as the two Moab servers access the same database/checkpoint file.

> As Moab uses timestamping in the lock file to implement high availability, the clocks on both servers require synchronization; all machines in a cluster must be synchronized to the same time server.

Moab high availability and Torque high availability operate independently of each other. If a job is submitted with *`msub`* and the primary Moab server is down, *`msub`* tries to connect to the fallback Moab server. Once the job is given to Torque, if Torque can't connect to the primary pbs_server, it tries to connect to the fallback pbs_server. For example:

- A job is submitted with *`msub`*, but Moab is down on `server01`, so *`msub`* contacts Moab running on `server02`.
- A job is submitted with *`msub`* and Moab hands it off to Torque, but pbs_server is down on `server01`, so *`qsub`* contacts pbs_server running on `server02`.

When you shut down or restart Moab on both servers, you must run the command twice. A single shutdown (mschedctl -k) or restart (mschedctl -R) command will go to the primary server and kill it, causing the secondary server to fall back and start operating. To kill the secondary server, resubmit the command.

> Do not use anything but a plain simple NFS fileshare that is not used by anybody or anything else (i.e., only Moab can use the fileshare).

Do not use any general-purpose NAS, do not use any parallel file system, and do not use company-wide shared infrastructure to set up Moab high availability using 'native' high availability.

When mounting a remote machine to the NFS server using the /etc/fstab file, do not use the option 'noatime'. This option disables access time updates on the file system that Moab relies on to prevent scheduling conflicts between multiple Moab servers.

18.2.2 Configuring High Availability on a Networked File System

Because the two Moab servers access the same files, configuration is only required in the `moab.cfg` file. The two hosts that run Moab must be configured with the `SERVER` and `FBSERVER` parameters. File lock is turned on using the FLAGS=filelockha flag. Specify the lock file with the `HALOCKFILE` parameter. The following example illustrates a possible configuration:

```
SCHEDCFG[Moab]  SERVER=host1:42559
SCHEDCFG[Moab]  FBSERVER=host2
SCHEDCFG[Moab]  FLAGS=filelockha
SCHEDCFG[Moab]  HALOCKFILE=/opt/moab/.moab_lock
```

Use the HALOCKUPDATETIME parameter to specify how frequently the primary server updates the timestamp on the lock file. Use the HALOCKCHECKTIME parameter to specify how frequently the secondary server checks the timestamp on the lock file.

```
HALOCKCHECKTIME 9
HALOCKUPDATETIME 3
```

In the preceding example, the secondary server checks the lock file for updates every 9 seconds. The `HALOCKUPDATETIME` parameter is set to 3 seconds, permitting the primary server three opportunities to update the timestamp for each time the secondary server checks the timestamp on the lock file.

`FBSERVER` does not take a port number. The primary server's port is used for both the primary server and the fallback server.

18.2.3 Confirming High Availability on a Networked File System

Admins can run the mdiag -S -v command to view which Moab server is currently scheduling and responding to client requests.

18.2.4 Other High Availability Configuration

Moab has many features to improve the availability of a cluster beyond the ability to automatically relocate to another execution server. The following table describes some of these features:

Feature	Description
AMCFG[] BACKUPHOST	If using the Moab Accounting Manager, you can enable high availability with the accounting manager by specifying a backup server as in the following example: `AMCFG[mam] BACKUPHOST=headnode2`
FBSERVER	If you are communicating to an HA configuration from the Moab Grid Control, you can use this parameter to point Moab to the fallback server.
JOBACTIONONNODEFAILURE	If a node allocated to an active job fails, it is possible for the job to continue running indefinitely even though the output it produces is of no value. Setting this parameter enables the scheduler to automatically preempt these jobs when a node failure is detected, possibly allowing the job to run elsewhere and also allowing other allocated nodes to be used by other jobs.
SCHEDCFG[] RECOVERYACTION	If a catastrophic failure event occurs (SIGSEGV or SIGKILL signal is triggered), Moab can be configured to automatically restart, trap the failure, ignore the failure, or behave in the default manner for the specified signal. These actions are specified using the values `RESTART`, `TRAP`, `IGNORE`, or `DIE`, as in the following example: `SCHEDCFG[bas] MODE=NORMAL RECOVERYACTION=RESTART`

SCHEDCFG[] <Failover Trigger Definition>	HA failover trigger.
	`SCHEDCFG[Moab]`
	`TRIGGER=atype=exec,etype=failure,action="/opt/moab/hafailover.sh"`

18.3 Malleable Jobs

Malleable jobs are jobs that can be adjusted in terms of resources and duration required, and which allow the scheduler to maximize job responsiveness by selecting a job's resource shape or footprint prior to job execution. Once a job has started, however, its resource footprint is fixed until job completion.

To enable malleable jobs, the underlying resource manager must support dynamic modification of resource requirements prior to execution (i.e., Torque) and the jobs must be submitted using the TRL (task request list) resource manager extension string. With the `TRL` attribute specified, Moab will attempt to select a start time and resource footprint to minimize job completion time and maximize overall effective system utilization (i.e., `<AverageJobEfficiency> * <AverageSystemUtilization>`).

Example 18-2: malleable jobs

```
> qsub -l nodes=1,trl=1@3600:2@1800:4@900 testjob.cmd
job 72436.orion submitted
```

Moab will execute the job in one of the following configurations: 1 node for 1 hour, 2 nodes for 30 minutes, or 4 nodes for 15 minutes.

18.4 Identity Managers

The Moab identity manager interface can be used to coordinate global and local information regarding users, groups, accounts, and classes associated with compute resources. The identity manager interface can also be used to allow Moab to automatically and dynamically create and modify user accounts and credential attributes according to current workload needs.

Only one identity manager can be configured at a time.

In this section:

18.4.1 Identity Manager Overview

Moab allows sites extensive flexibility when it comes to defining credential access, attributes, and relationships. In most cases, use of the USERCFG, GROUPCFG, ACCOUNTCFG, CLASSCFG, and QOSCFG parameters is adequate to specify the needed configuration. However, in certain cases such as the following, this approach might not be ideal or even adequate:

- Environments with very large user sets
- Environments with very dynamic credential configurations in terms of fairshare targets, priorities, service access constraints, and credential relationships
- Grid environments with external credential mapping information services
- Enterprise environments with fairness policies based on multi-cluster usage

Moab addresses these and similar issues through the use of an identity manager. An identity manager is configured with the IDCFG parameter and enables Moab to exchange information with an external identity management service. As with Moab resource manager interfaces, this service can be a full commercial package designed for this purpose, or something far simpler such as a web service, text file, or database.

18.4.2 Basic Configuration

Configuring an identity manager in basic read-only mode can be accomplished by simply setting the `SERVER` attribute. If Moab is to interact with the identity manager in read/write mode, some additional configuration might be required.

BLOCKCREDLIST	
Format	One or more of these comma-delimited object types: `acct`, `group`, or `user`.

BLOCKCREDLIST

Details

If specified, Moab will block all jobs associated with credentials not explicitly reported in the most recent identity manager update. If the credential appears on subsequent updates, resource access will be immediately restored.

Jobs will only be blocked if fairshare is enabled. This can be accomplished by setting the FSPOLICY parameter to any value such as in the following example:

```
FSPOLICY DEDICATEDPS
```

Example

```
IDCFG[test01] BLOCKCREDLIST=acct,user,groups
```

Moab will block any jobs associated with accounts, users, or groups not in the most recent identity manager update.

CREATECRED

Format	`<BOOLEAN>` (default is `FALSE`)
Details	Specifies whether Moab should create credentials reported by the identity manager that have not yet been locally discovered or loaded via the resource manager. By default, Moab will only load information for credentials that have been discovered outside of the identity manager.
Example	`IDCFG[test01] CREATECRED=TRUE` *Moab will create credentials from* `test01` *that have not been previously loaded.*

REFRESHPERIOD

Format

[[`DD:`]`HH:`]`MM:`]`SS` or `INFINITY` (default is `INFINITY`)

Note: The former values of MINUTE, HOUR, DAY or NONE are deprecated and may be removed in a future release.

Details

Moab will refresh the identity manager information on the specified period relative to the scheduler start time. If `INFINITY` is specified, the information is updated only at Moab start up.

Example

```
IDCFG[test01] REFRESHPERIOD=4:00:00
```

REFRESHPERIOD	
	Moab queries the identity manager every four hours.

REQUIREDUSERLIST	
Format	One or more comma-delimited object types from the `user` list.
Details	Lets you dynamically set the user list with a class for jobs. Removing a user from the REQUIREDUSERLIST will not affect the user's running jobs. However, the user's idle jobs will become blocked because the user no longer has access to the class requested.
Example	`IDCFG[test01] CLASS:<name> REQUIREDUSERLIST=<user>`

RESETCREDLIST	
Format	One or more of these comma-delimited object types: `acct`, `group`, or `user`.
Details	If specified, Moab will reset the account access list and fairshare cap and target for all credentials of the specified type(s) regardless of whether they are included in the current info manager report. Moab will then load information for the specified credentials.
Example	`IDCFG[test01] RESETCREDLIST=group` *Moab will reset the account access list and fairshare target for all groups.*

SERVER	
Format	`<URL>`
Details	Specifies the protocol/interface to use to contact the identity manager.
Example	`IDCFG[test01] SERVER=exec://$HOME/example.pl` *Moab will use `example.pl` to communicate with the identity manager.*

SERVER	

UPDATEREFRESHONFAILURE	
Format	`<BOOLEAN>` (default is `FALSE`)
Details	When an `IDCFG` script fails, it retries almost immediately and continuously until it succeeds. When `UPDATEREFRESHONFAILURE` is set to `TRUE`, a failed script does not attempt to rerun immediately, but instead follows the specified `REFRESHPERIOD` schedule. When set to `TRUE`, `UPDATEREFRESHONFAILURE` updates the script execution timestamp, even if the script does not end successfully.
Example	`IDCFG[info] SERVER=exec:///home/tshaw/test/1447/bad_script.pl REFRESHPERIOD=hour UPDATEREFRESHONFAILURE=TRUE`

18.4.3 Importing Credential Fairness Policies

One common use for an identity manager is to import fairness data from a global external information service. As an example, assume a site needed to coordinate Moab group level fairshare targets with an allocation database that constrains total allocations available to any given group. To enable this, a configuration like the following might be used:

```
IDCFG[alloc] SERVER=exec://$TOOLSDIR/idquery.pl
...
```

The `tools/idquery.pl` script could be set up to query a local database and report its results to Moab. Each iteration, Moab will then import this information, adjust its internal configuration, and immediately respect the new fairness policies.

18.4.4 Identity Manager Data Format

When an identity manager outputs credential information either through an `exec` or `file` based interface, the data should be organized in the following format:
`<CREDTYPE>:<CREDID> <ATTR>=<VALUE>`

Where:

- `<CREDTYPE>` is one of `user`, `group`, `account`, `class`, or `qos`.
- `<CREDID>` is the name of the credential.

- `<ATTR>` is one of `adminlevel`, `alist`, `chargerate`, `comment`, `emailaddress`, `fstarget`, `globalfstarget`, `globalfsusage`, `maxjob`, `maxmem`, `maxnode`, `maxpe`, `maxproc`, `maxps`, `maxwc`, `MAX.WCLIMIT`, `plist`, `priority`, `qlist`, or `role`. Multi-dimensional policies work here also.
- `<VALUE>` is the value for the specified attribute.

To clear a comment, set its value to `""`; for example: `comment=""`.

The following output may be generated by an `exec` based identity manager:

```
group:financial fstarget=16.3 alist=project2
group:marketing fstarget=2.5
group:engineering fstarget=36.7
group:dm fstarget=42.5
user:jason adminlevel=3
account:sales maxnode=128 maxjob=8,16
```

The following example limits user `bob` to 8 `matlab` generic resources:

```
user:bob MAXGRES[matlab]=8
```

To specify unlimited use of generic resources, set the value to `-1`.

18.4.5 Identity Manager Conflicts

When local credential configuration (as specified via `moab.cfg`) conflicts with identity manager configuration, the identity manager value takes precedence and the local values are overwritten.

18.4.6 Refreshing Identity Manager Data

By default, Moab only loads identity manager information once when it is first started up. If the identity manager data is dynamic, then you might want Moab to periodically update its information. To do this, set the `REFRESHPERIOD` attribute of the `IDCFG` parameter. Values are documented in the following table:

Value	Description
minute	Update identity information once per minute.
hour	Update identity information once per hour.

Value	Description
day	Update identity information once per day.
infinity	Update identity information only at start-up (default).

Example 18-3: REFRESHPERIOD

```
IDCFG[hq] SERVER=exec://$TOOLSDIR/updatepolicy.sh REFRESHPERIOD=hour
```

Job credential feasibility is evaluated at job submission and start time.

Related Topics

- 2.7 Credentials
- 5.2 Usage Limits/Throttling Policies

18.5 Generic System Jobs

Generic system jobs are system jobs with a trigger. They are useful for specifying steps in a workflow.

In this section:

18.5.1 Creating a Generic System Job

Generic system jobs are specified via a job template. The template can be selectable and you must use the `GENERICSYSJOB` attribute to let Moab know that this job template describes a generic system job and to specify a trigger, as shown in the following example:

```
JOBCFG[gen]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/genericTrig.py",Timeout=5:00
```

The Trigger

The generic system job's trigger that meets certain criteria. This trigger must have a timeout, an `Atype=Exec`, and the `EType` must equal 'start'. The timeout of the trigger will be used as the walltime for the job. The trigger will begin when the system job begins and the job will be considered completed when the trigger completes. The job will have the same completion code as the trigger. The walltime on the job template is not applicable in this case since the timeout of the trigger will be the walltime.

If the trigger fails, an error message will be attached to all of the job's parent VCs. You can view this in the `--xml` output of the VC query. The message includes the location of STDIN, STDOUT, and STDERR files. For example:

```
mvcctl -q ALL --xml

<Data>
<vc CREATETIME="1320184350" DESCRIPTION="Moab.1"
  FLAGS="DESTROYOBJECTS,DESTROYWHENEMPTY,HASSTARTED,WORKFLOW"
  JOBS="Moab.1" NAME="vc1" OWNER="user:frank">
<ACL aff="positive" cmp="%=" name="frank" type="USER"></ACL>
<MESSAGES>
<message COUNT="1" CTIME="1320184362"
     DATA="Trigger 10 failed on job Moab.1.setup- STDIN:
/tmp/ByLLl2wv/spool/vm.py.ieWPPS5 STDOUT:
/tmp/ByLLl2wv/spool/vm.py.oDMIXAW STDERR /tmp/ByLLl2wv/spool/vm.py.e2jD5iN"
     EXPIRETIME="1322776362" OWNER="frank" PRIORITY="0"
     TYPE="other" index="0"></message>
</MESSAGES>
<Variables>
<Variable name="VMID">vm1</Variable>
<Variable name="HV">TRUE</Variable>
</Variables>
</vc>
</Data>
```

You can specify other triggers on a generic system job using the `TRIGGER` attribute and delimiting them with semicolons. For example:

```
JOBCFG[gen]  GENERICSYSJOB=<genericSystemJobTriggerSpecs>
JOBCFG[gen]  TRIGGER=<triggerSpecs>;TRIGGER=<triggerSpecs>
```

18.5.2 Workflows Using Job Template Dependencies

To create workflows, use the following format:

```
JOBCFG[gen]  TEMPLATEDEPEND=AFTERANY:otherTemplate
```

This will create a job based on the template `otherTemplate`. The generic job will run after the otherTemplate job has finished. Afterany in the example means after all other jobs have completed, regardless of success.

Inheriting Resources in Workflows

The `INHERITRES` flag can be used to cause the same resources in one step of a workflow to be passed to the next step:

```
JOBCFG[gen]  TEMPLATEDEPEND=AFTERANY:otherTemplate
JOBCFG[otherTemplate]  INHERITRES=TRUE
```

This example forces the job based on `otherTemplate` to have the same resource requirements as its parent. When the `otherTemplate` job is finished, the `INHERITRES` flag will cause the parent to run on the same resources as the child.

The job that finishes first will pass its allocation up.

Any variables on the original job will be passed to the other jobs in the workflow. Variables can be added by other jobs in the workflow via the sets attribute in the generic system job's trigger. Other triggers must then request that variable name in the command line options.

You will need to set the carat (^) in order for the variable to be sent up to the job group.

If you set the variable, you need to set it in the STDOUT of the trigger script. See the example below:

```
JOBCFG[W1] GENERICSYSJOB=...,action='$HOME/W1.py $ipaddress' TEMPLATEDEPEND=AFTER:W2
JOBCFG[W2] TRIGGER=...,action='$HOME/W2.py',sets=^ipaddress
```

If a variable value is not set in STDOUT, it will be set to `TRUE`.

To set the variable to a specific value, the `W2.py` script must set the value in its STDOUT:

```
print "ipaddress=10.10.10.1" #This will be parsed by Moab and set as the value of the
"ipaddress" variable
```

Example 18-4: CreateVM

To create a VM with a workflow using job template dependencies and generic system jobs, use the following format:

```
#The job template that is "gate" to the workflow
JOBCFG[CreateVMWithSoftware] TEMPLATEDEPEND=AFTEROK:InstallSoftware SELECT=TRUE

JOBCFG[InstallSoftware]
GENERICSYSJOB=EType=start,AType=exec,Action="$HOME/setupSoftware.py
$IPAddr",Timeout=30:00
JOBCFG[InstallSoftware] INHERITRES=TRUE
JOBCFG[InstallSoftware] TEMPLATEDEPEND=AFTEROK:CreateVM

JOBCFG[CreateVM] GENERICSYSJOB=EType=start,AType=exec,Action=$HOME/installVM.py
$HOSTLIST",Timeout=1:00:00,sets=^IPAddr
JOBCFG[CreateVM] INHERITRES=TRUE
```

The user will then submit the job requesting what they need in the VM:

```
msub -1
walltime=2:00:00,template=CreateVMWithSoftware,nodes=1:ppn=4,mem=1024ActualWorkload.py
```

The job will have the CreateVMWithSoftware template applied to it and will create the InstallSoftware job. The InstallSoftware job, because of INHERITRES, will have the same resource request (4 procs, 1GB of memory). This job then has its template applied to it which will do the same thing in creating the CreateVM job. The CreateVM job will then run, the trigger script will return the IP address of the new VM and pass its allocation up to the InstallSoftware job. The InstallSoftware job will use the IPAddr variable to find the VM and install the software. It will then return its resources up to the parent job, which will run the actual workload.

Chapter 19: Database Configuration

Moab supports connecting to a database via native SQLite3, and it can also connect to other databases using the ODBC driver. These optional external databases store some additional information that the MongoDB database does not and allow you to query them directly using SQL. These databases are slower, however, and only SQLite3, which does not allow external queries, is supported.

The SQLite3 connection is for storing statistics. Consider reviewing the SQLite web page Appropriate Uses for SQLite for information regarding the suitability of using SQLite3 on your system.

Connecting to an external database makes Moab more searchable, allowing you to run queries for statistics and events rather than using regular expressions to draw the information from the Moab flat files.

Moab must use an ODBC-compliant database to report statistics with Viewpoint reports.

In this chapter:

19.1 SQLite3

Moab supports connecting to a database via native SQLite3. Database installation and configuration occurs automatically during normal Moab installation (configure, make install). If you did not follow the normal process to install Moab and need to install the database, do the following to manually install and configure Moab database support:

1. Create the database file `moab.db` in your moab home directory by running the following command from the root of your unzipped Moab build directory: `perl buildutils/install.sqlite3.pl <moab-home-directory>`

- Verify that the command worked by running `lib/sqlite3 <moab-home-directory>/moab.db`; at the resulting prompt, type `.tables` and press `ENTER`. You should see several tables such as `mcheckpoint` listed. Exit from this program with the `.quit` command.
- The `perl buildutils/install.sqlite3.pl <moab-home-directory>` command could fail if your operating system cannot find the SQLite3 libraries. Also, Moab fails if unable to identify the libraries. To temporarily force the libraries to be found, run the following command: `export LD_LIBRARY_PATH=<location where libraries were copied>`

2. In the `moab.cfg` file in the `etc/` folder of the `home` directory, add the following line:

```
USEDATABASE  INTERNAL
```

To verify that Moab is running with SQLite3 support, start Moab and run the mdiag -S -v command. If there are no database-related error messages displayed, then Moab should be successfully connected to a database.

> `moabd` is a safe and recommended method of starting Moab if things are not installed in their default locations.

19.2 Connecting to a MySQL Database with an ODBC Driver

This section shows how to set up and configure Moab to connect to a MySQL database using the MySQL ODBC driver, which assumes the necessary MySQL and ODBC drivers have already been installed and configured.

To set up and configure Moab to connect to a MySQL database using the MySQL ODBC driver, do the following.

This solution has been tested and works with these versions:

- libmyodbc - 5.1.5
- MySQL 5.1

1. Download and install Moab. Configure Moab as normal but add the following in the Moab configuration file (`moab.cfg`):

```
USEDATABASE             ODBC
```

```
# Turn on stat profiling
USERCFG[DEFAULT]        ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]       ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]         ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]       ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]     ENABLEPROFILING=TRUE
NODECFG[DEFAULT]        ENABLEPROFILING=TRUE
```

2. Create the database in MySQL using the MySQL database dump contained in the `moab-db-mysql-create.sql` file. This file is located in the `contrib/sql` directory.

   This contrib/sql directory is in the expanded tarball directory.

   Run the following command:

   ```
   mysql -u root -p < moab-db-mysql-create.sql
   ```

3. Configure the MySQL and ODBC driver. The `odbcinst.ini` file must be contained in `/etc`.

   Run the following command to find the MySQL ODBC client driver. You could also query the `libmyodbc` package that was installed.

   ```
   [root]# updatedb
   [root]# locate libmyodbc
   ```

   ```
   [MySQL]
   Description = ODBC for MySQL
   Driver = /usr/lib/odbc/libmyodbc.so
   ```

4. Configure Moab to use the MySQL ODBC driver. Moab uses an ODBC datastore file to connect to MySQL using ODBC. This file must be located in the Moab home directory (`/opt/moab` by default) and be named `dsninfo.dsn`, which is used by Moab. You need to have the following data in both `/etc/odbc.ini` and `$MOABHOMEDIR/dsninfo.dsn`:

   ```
   [ODBC]
   Driver = MySQL
   USER = <username>
   PASSWORD = <password>
   Server = localhost
   Database = Moab
   Port = 3306
   ```

   The user should have read/write privileges on the Moab database.

   The preceding example file tells ODBC to use the MySQL driver, username `<username>`, password `<password>`, and to connect to MySQL running on the localhost on port `3306`. ODBC uses this information to connect to the database called `Moab`.

5. Test the ODBC to MySQL connection by running the *`isql`* command, which reads the `/etc/odbc.ini` file:

```
$ isql -v ODBC
+---------------------------------------+
| Connected! |
| |
| sql-statement |
| help [tablename] |
| quit |
| |
+---------------------------------------+
SQL> show tables;
+-------------------------------------------------------------------+
| Tables_in_Moab |
+-------------------------------------------------------------------+
| EventType |
| Events |
| GeneralStats |
| GenericMetrics |
| Moab |
| NodeStats |
| NodeStatsGenericResources |
| ObjectType |
| mcheckpoint |
+-------------------------------------------------------------------+
SQLRowCount returns 10
10 rows fetched
SQL>
```

If you encounter any errors using the *`isql`* command, there was a problem setting up the ODBC to MySQL connection. Try the following debugging steps to resolve the issue:

a. The `odbcinst.ini` and `odbc.ini` files are usually assumed to be located in `/etc`, but that is not always true. Use the `odbcinst -j` command to determine the assumed location of the files in your configuration.

```
[root#] odbcinst -j
unixODBC 2.2.12
DRIVERS............: /etc/unixODBC/odbcinst.ini
SYSTEM DATA SOURCES: /etc/unixODBC/odbc.ini
USER DATA SOURCES..: /home/adaptive/.odbc.ini
```

b. Because `odbcinst.ini` and `odbc.ini` are expected in `/etc/unixODBC`, not `/etc`, move them from `/etc` to `/etc/unixODBC`.

c. Use the `strace` command to determine where `isql` expects the `odbc.ini` and `odbcinst.ini` files. Note the location where `isql` expects these files.

```
$ strace isql -v ODBC
```

6. With the ODBC driver configured, the database created, and Moab configured to use the database, start Moab for it to begin storing information in the created database.

> `> moabd` is a safe and recommended method of starting Moab if things are not installed in their default locations.

Related Topics

- 19.6 Importing Statistics from stats/DAY.* to the Moab Database

19.3 Connecting to a PostgreSQL Database with an ODBC Driver

This section shows how to set up and configure Moab to connect to a PostgreSQL database using the ODBC driver, which assumes the necessary ODBC drivers have already been installed and configured.

> Occasionally vacuuming your PostgreSQL database could improve Moab performance. See the PostgreSQL documentation for information on how to vacuum your database.

To set up and configure Moab to connect to a PostgreSQL database using the ODBC driver, do the following:

> This solution has been tested and works with the file version odbc-postgresql - 1:08.03.0200-1.2

1. Configure the PostgreSQL and ODBC driver. `odbcinst.ini` file must be contained in `/etc`.

   > Run the following commands to find the PostgreSQL ODBC client driver and setup file. You could also query the `libodbcpsql` package that was installed.
   >
   > ```
   > [root]# updatedb
   > [root]# locate psqlodbc
   > [root]# locate libodbcpsql
   > ```

   ```
   [PostgreSQL]
   Description = PostgreSQL ODBC driver
   Driver = /usr/lib/odbc/psqlodbca.so
   Setup = /usr/lib/odbc/libodbcpsqlS.so
   Debug = 0
   CommLog = 1
   ```

```
UsageCount = 2
```

2. Configure Moab to use the PostgreSQL ODBC driver. Moab uses an ODBC datastore file to connect to PostgreSQL using ODBC. This file must be located in the Moab home directory (`/opt/moab` by default) and be named `dsninfo.dsn`, which is used by Moab. If the following content, which follows the standard ODBC driver file syntax, is not already included in the `/etc/odbc.ini` file, make sure that you include it. Also, include the same content in the `dsninfo.dsn` file.

   ```
   [ODBC]
   Driver = PostgreSQL
   Description = PostgreSQL Data Source
   Servername = localhost
   Port = 5432
   Protocol = 8.4
   UserName = postgres
   Password = moab
   Database = Moab
   ```

   The user should have read/write privileges on the Moab database.

   The preceding example file tells ODBC to use the PostgreSQL driver, `postgres` user, `moab` password, and to connect to PostgreSQL running on the localhost on port 5432. ODBC uses this information and connects to the database called `Moab`.

3. Test the ODBC to PostgreSQL connection by running the *`isql`* command, which reads the `/etc/odbc.ini` file. If connected, you should be able to run the *`help`* command.

   If you encounter any errors using the *`isql`* command, there was a problem setting up the ODBC to MySQL connection. Try the following debugging steps to resolve the issue:

   a. The `odbcinst.ini` and `odbc.ini` files are usually assumed to be located in `/etc`, but that is not always true. Use the `odbcinst -j` command to determine the assumed location of the files in your configuration.

      ```
      [root#] odbcinst -j
      unixODBC 2.2.12
      DRIVERS............: /etc/unixODBC/odbcinst.ini
      SYSTEM DATA SOURCES: /etc/unixODBC/odbc.ini
      USER DATA SOURCES..: /home/adaptive/.odbc.ini
      ```

   b. Because `odbcinst.ini` and `odbc.ini` are expected in `/etc/unixODBC`, not `/etc`, move them from `/etc` to `/etc/unixODBC`.

   c. Use the `strace` command to determine where `isql` expects the `odbc.ini` and `odbcinst.ini` files. Note the location where `isql` expects these files.

      ```
      $ strace isql -v ODBC
      ```

4. Create the database in PostgreSQL using the `moab-db-postgresql.sh` setup script contained in the `contrib/sql` directory.

This contrib/sql directory is in the expanded tarball directory.

- Run the script and provide the DB username that will attach to the Moab database (you must supply a DB username or the script will exit). The default admin user is `postgres`, but you can make a new user at this time:

  ```
  > ./moab-db-postgresql.sh postgres
  Create db user "postgres" in postgreSQL? (y/n)>
  ```

- The script asks if you want to create the DB user you specified in postgreSQL. If the DB user already exists, answer 'n'. Otherwise, the DB user is created and it asks for the new user's password.
- The script then creates the database 'Moab'.
- Finally, as the DB user you provided, the script imports the DB schema from `moab-db-postgresql-create.sql` into the Moab database.

5. Download and install Moab. Install and configure Moab as normal but add the following in the Moab configuration file (`moab.cfg`):

```
USEDATABASE              ODBC
# Turn on stat profiling
USERCFG[DEFAULT]         ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]        ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]          ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]        ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]      ENABLEPROFILING=TRUE
NODECFG[DEFAULT]         ENABLEPROFILING=TRUE
```

6. With the ODBC driver configured, the database created, and Moab configured to use the database, start Moab for it to begin storing information in the created database.

> moabd is a safe and recommended method of starting Moab if things are not installed in their default locations.

Related Topics

- 19.6 Importing Statistics from stats/DAY.* to the Moab Database

19.4 Connecting to an Oracle Database with an ODBC Driver

This section shows how to set up and configure Moab to connect to an Oracle database using the ODBC driver.

In this section:

19.4.1 To Connect to an Oracle Database with an ODBC Driver

1. Install and configure the Oracle Instant Client with ODBC supporting libraries. For instructions, see the section Installing the Oracle Instant Client below.
2. Open your Moab configuration file (`$MOABHOMEDIR/moab.cfg`) and add the following lines to the end of the file:

```
USEDATABASE ODBC

# Turn on stat profiling
USERCFG[DEFAULT]         ENABLEPROFILING=TRUE
GROUPCFG[DEFAULT]        ENABLEPROFILING=TRUE
QOSCFG[DEFAULT]          ENABLEPROFILING=TRUE
CLASSCFG[DEFAULT]        ENABLEPROFILING=TRUE
ACCOUNTCFG[DEFAULT]      ENABLEPROFILING=TRUE
NODECFG[DEFAULT]         ENABLEPROFILING=TRUE
```

3. Configure the Oracle ODBC Driver. The `odbcinst.ini` file must be contained in `/etc`.

```
[root]# vim /etc/odbcinst.ini
```

Run the following command to find the Oracle Instant Client driver. You could also query the Oracle Instant Client package that was installed.

```
[root]# updatedb && locate libsqora
```

Add the following text to the file:

```
[Oracle 11g ODBC driver]
Description     = Oracle ODBC driver for Oracle 11g
Driver          = /usr/lib/oracle/11.2/client64/lib/libsqora.so.11.1
Setup           =
FileUsage       =
CPTimeout       =
CPReuse         =
Driver Logging  = 7

[ODBC]
Trace = Yes
TraceFile = /tmp/odbc.log
ForceTrace = Yes
Pooling = No
```

```
DEBUG = 1
```

`Driver Logging` is set high (level 7) so that you can debug during the installation and configuration process if necessary. You can decrease the setting or remove the directive once you finish the process.

To configure the location of the ODBC log (`/tmp/odbc.log`), set the `TraceFile` attribute shown in the example above. See unixODBC without the GUI on the unixODBC website for more information.

4. Because the driver installed in step 1 is a shared library, run `ldd` to verify that it and all of its dependencies are installed and working:

```
[root]# ldd /usr/lib/oracle/11.2/client64/lib/libsqora.so.11.1
            linux-vdso.so.1 =>  (0x00007fff631ff000)
            libdl.so.2 => /lib64/libdl.so.2 (0x00007f8afbe83000)
            libm.so.6 => /lib64/libm.so.6 (0x00007f8afbbff000)
            libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f8afb9e1000)
            libnsl.so.1 => /lib64/libnsl.so.1 (0x00007f8afb7c8000)
            libclntsh.so.11.1 =>
/usr/lib/oracle/11.2/client64/lib/libclntsh.so.11.1 (0x00007f8af8e59000)
            libodbcinst.so.1 => not found
            libc.so.6 => /lib64/libc.so.6 (0x00007f8af8ac5000)
            /lib64/ld-linux-x86-64.so.2 (0x0000003bdb000000)
            libnnz11.so => /usr/lib/oracle/11.2/client64/lib/libnnz11.so
(0x00007f8af86f8000)
            libaio.so.1 => /lib64/libaio.so.1 (0x00007f8af84f6000)
```

If the command returns `libodbcinst.so.1 => not found`, create a symbolic link from `/usr/lib64/libodbcinst.so.1` to `/usr/lib64/libodbcinst.so.2`. This is a known Red Hat issue. See Red Hat Bugzilla for more information.

```
[root]# locate libodbcinst

/usr/local/lib/libodbcinst.so.2

[root]# cd /usr/lib64
[root]# ln -s libodbcinst.so.2 libodbcinst.so.1
```

Rerun `ldd`. It should load `libsqora.so.11.1` without error, as shown in the `ldd` example above.

If the `ldd` command returns a warning like this: "ldd: warning: you do not have execution permission for `/usr/lib/oracle/11.2/client64/lib/libsqora.so.11.1'", run the following command:

```
[root]# chmod 755 /usr/lib/oracle/11.2/client64/lib/lib*
```

Rerun `ldd`. It should load `libsqora.so.11.1` without error, as shown in the `ldd` example above.

5. Configure Moab to use the Oracle ODBC driver. This example assumes that a Moab user exists and has been granted read and write privileges to the MOAB database instance referred to on the page.

```
[root]# vim $MOABHOMEDIR/dsninfo.dsn
```

Add the following lines to the file, but change `ServerName`, `UserName`, and `Password` to suit your own system. `ServerName` is the name of the Oracle database instance. `Username` and `Password` are the credentials used to connect to that instance.

```
[ODBC]
Application Attributes = T
Attributes = W
BatchAutocommitMode = IfAllSuccessful
BindAsFLOAT = F
CloseCursor = F
DisableDPM = F
DisableMTS = T
Driver = Oracle 11g ODBC driver
DSN = ODBC
EXECSchemaOpt =
EXECSyntax = T
Failover = T
FailoverDelay = 10
FailoverRetryCount = 10
FetchBufferSize = 64000
ForceWCHAR = F
Lobs = T
Longs = T
MaxLargeData = 0
MetadataIdDefault = F
QueryTimeout = T
ResultSets = T
ServerName = MOAB
SQLGetData extensions = F
Translation DLL =
Translation Option = 0
DisableRULEHint = T
UserID = moab
Password = moab
StatementCache=F
CacheBufferSize=20
UseOCIDescribeAny=F
MaxTokenSize=8192
```

6. Add the contents of the `dsninfo.dsn` file to `/etc/odbc.ini`. Because the contents of `dsninfo.dsn` are required in both files, use the following command to concatenate

the contents of `dsninfo.dsn` to `/etc/odbc.ini`. If the `odbc.ini` file already has content, verify that there are no conflicts.

```
[root]# cat $MOABHOMDIR/dsninfo.dsn >> /etc/odbc.ini
```

7. Create a directory to store the `tnsnames.ora` file you will create in the next step:

```
[root]# mkdir /etc/oracle
```

8. Create the `tnsnames.ora` file. The `ServerName` in `$MOABHOMEDIR/dsninfo.dsn` tells the Oracle ODBC driver what `tnsnames.ora` entry to use (`MOAB`). The `MOAB` `tnsnames` entry tells the Oracle ODBC driver to connect to server `adaptive-oracle` on the local domain (`ac`) on port 1561 using TCP and to connect to the Oracle instance named `MOAB` (the SID is the unique name of the instance).

```
[root]# cat >/etc/oracle/tnsnames.ora <<EOL
MOAB =
   (DESCRIPTION =
     (ADDRESS_LIST =
       (ADDRESS = (PROTOCOL = TCP)(HOST = adaptive-oracle)(PORT = 1561))
     )
     (CONNECT_DATA =
       (SID = MOAB)
     )
   )
EOL
```

9. Create a profile script (`oracle-instant-client.sh`) to be invoked by the operating system at startup. This script will set the `ORACLE_HOME`, `TWO_TASK`, and `TNS_ADMIN` environment variables required by Oracle and will amend the `LD_LIBRARY_PATH` to include required Oracle client libraries in the library search path.

```
[root]# cat >/etc/profile.d/oracle-instant-client.sh <<EOL
# Set ORACLE_HOME to the directory where the bin and lib directories are located
for the oracle client
export ORACLE_HOME=/usr/lib/oracle/11.2/client64

# No need to add ORACLE_HOME to the linker search path. oracle-instant-client.conf
in
# /etc/ld.so.conf.d should already contain /usr/lib/oracle/11.2/client64.
# Alternatively, you can set it here by uncommenting the following line:
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib

# Define the default location where Oracle should look for the server
export TWO_TASK=//adaptive-oracle:1561/listener

# Define where to find the tnsnames.ora file
export TNS_ADMIN=/etc/oracle
EOL
```

10. Source the `oracle-instant-client.sh` script and verify that each environment variable is set correctly:

```
[root]# source /etc/profile.d/oracle-instant-client.sh
```

```
[root]# echo $ORACLE_HOME
[root]# echo $LD_LIBRARY_PATH
[root]# echo $TWO_TASK
[root]# echo $TNS_ADMIN
```

11. Modify either the Moab startup script (`/etc/init.d/moab`) – recommended – or the `moabd` script (`/opt/moab/sbin/moabd`) to source `oracle-instant-client.sh`.

    - Moab startup script (recommended): the following example suggests a location to source the `oracle-instant-client.sh` script within the Moab startup script.

    ```
    ...

    # Export all environment variables required by the Oracle Instant Client
    . /etc/profile.d/oracle-instant-client.sh

    export MOABHOMEDIR=/opt/moab

    ...
    ```

    - `moabd` shell script: the following example will resemble the `moabd` script in `/opt/moab/sbin`. Note that the `moabd` script is not invoked by the Moab startup script; The Moab startup script invokes the Moab binary (`/opt/moab/sbin/moab`) by default.

    ```
    #!/bin/sh
    #
    #   Copyright (C) 2025 by Adaptive Computing Enterprises, Inc. All Rights
    Reserved.
    #

    # Export all environment variables required by the Oracle Instant Client
    . /etc/profile.d/oracle-instant-client.sh

    MOABHOMEDIR="/opt/moab" LD_LIBRARY_PATH="/opt/moab/lib:$LD_LIBRARY_PATH" moab
    "$@"
    ```

12. Verify the Oracle ODBC driver is working:

    ```
    isql -v ODBC
    +---------------------------------------+
    | Connected!                            |
    |                                       |
    | sql-statement                         |
    | help [tablename]                      |
    | quit                                  |
    |                                       |
    +---------------------------------------+
    ```

    If you encounter any errors using the *`isql`* command, there was a problem setting up the ODBC to Oracle connection. Try the following debugging steps to resolve the issue:

    a. The `odbcinst.ini` and `odbc.ini` files are usually assumed to be located in `/etc`, but that is not always true. Use the `odbcinst -j` command to determine the assumed location of the files in your configuration.

```
[root#] odbcinst -j
unixODBC 2.2.12
DRIVERS............: /etc/unixODBC/odbcinst.ini
SYSTEM DATA SOURCES: /etc/unixODBC/odbc.ini
USER DATA SOURCES..: /home/adaptive/.odbc.ini
```

b. Because `odbcinst.ini` and `odbc.ini` are expected in `/etc/unixODBC`, not `/etc`, move them from `/etc` to `/etc/unixODBC`.

c. Use the `strace` command to determine where `isql` expects the `odbc.ini` and `odbcinst.ini` files. Note the location where `isql` expects these files.

```
$ strace isql -v ODBC
```

13. If you have not already done so, create the database tables in Oracle using the `moab-db-oracle-create.sql` script located in the `contrib/sql` directory.

This contrib/sql directory is in the expanded tarball directory.

This example assumes that you are logged into the MOAB database instance (referred to on the page) as a Moab user with read and write privileges:

```
SQL> @./contrib/sql/moab-db-oracle-create.sql
```

14. Verify that the database schema installed correctly by listing the tables. Your results should look like this:

```
SQL> select table_name from all_tables where owner = 'MOAB';
+--------------------------------+
| TABLE_NAME                     |
+--------------------------------+
| TRIGGERS                       |
| MOAB                           |
| OBJECTTYPE                     |
| VCS                            |
| EVENTTYPE                      |
| JOBHISTORY                     |
| MCHECKPOINT                    |
| NODES                          |
| EVENTS                         |
| NODESTATSGENERICRESOURCES      |
| JOBS                           |
| RESERVATIONS                   |
| GENERICMETRICS                 |
| REQUESTS                       |
| GENERALSTATS                   |
| NODESTATS                      |
+--------------------------------+
SQLRowCount returns -1
16 rows fetched
```

15. Restart Moab:

```
[root]# mschedctl -R
```

16. Verify Moab is correctly configured to write to the Oracle database by doing each of the following steps:

    a. Tail the `moab.log` file for ODBC errors:

    ```
    # Check the $MOABHOMEDIR/log/moab.log file for ODBC errors. You should see a few
    hits even if there are no errors.
    [root]# tail -f $MOABHOMEDIR/log/moab.log | grep -i odbc
    ```

    b. Log in to the Moab Oracle database.

    In the first example below, `isql` will search `/etc/odbc.ini` for "[ODBC]". unixODBC will then use the Oracle 11g ODBC driver defined in `/etc/odbcinst.ini` to establish a connection. The `ServerName` in `/etc/odbc.ini` tells the Oracle driver to reference the `MOAB` `tnsnames` entry in `/etc/oracle/tnsnames.ora` for connection parameters.

    The second example uses `sqlplus` and a connect string to connect.

    Try both connection methods:

    ```
    # Log in to Oracle. Try both isql and sqlplus64 clients.
    [root]# isql -v ODBC
    ```

    ```
    [root]# sqlplus64 moab/moab@adaptive-oracle:1561/MOAB
    ```

    c. Select some data from one or more of the tables (Nodes, Events, and the like) to verify that data is being stored in the Moab Oracle instance:

    ```
    # sqlplus64 moab/moab@adaptive-oracle:1561/MOAB

    SQL*Plus: Release 11.2.0.4.0 Production on Fri Oct 4 14:59:02 2013

    Copyright (c) 1982, 2013, Oracle.  All rights reserved.


    Connected to:
    Oracle Database 11g Release 11.2.0.1.0 - 64bit Production

    SQL> select table_name from user_tables;

    TABLE_NAME
    ------------------------------
    JOBS
    REQUESTS
    RESERVATIONS
    VCS
    EVENTTYPE
    GENERALSTATS
    GENERICMETRICS
    NODESTATS
    NODESTATSGENERICRESOURCES
    EVENTS
    JOBHISTORY
    MCHECKPOINT
    NODES
    TRIGGERS
    MOAB
    ```

```
OBJECTTYPE

16 rows selected.
```

19.4.2 Installing the Oracle Instant Client

The following procedure demonstrates how to install the correct ODBC drivers for your Oracle database. This guide is a prerequisite for the Connecting to an Oracle Database with an ODBC Driver task. Each step must be performed as root.

1. Go to the Install Client Downloads page on the Oracle website. Choose the link that matches your system type (for instance, Instant Client for Linux x86-64). Choose `Accept License Agreement` at the top of the page and download the following RPM or zip files for your target version (such as 11.2):

   > The process of connecting Oracle to Moab Workload Manager has been tested on Oracle Instant Client version 11.2. The process might work with other versions, but they are not supported.

   - Basic (`oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm`)
   - SQL Plus (`oracle-instantclient11.2-sqlplus-11.2.0.4.0-1.x86_64.rpm`)
   - ODBC (`oracle-instantclient11.2-odbc-11.2.0.4.0-1.x86_64.rpm`)

2. Install the packages. This example installs the RPMs:

```
[root]# rpm -i ./oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm
[root]# rpm -i ./oracle-instantclient11.2-sqlplus-11.2.0.4.0-1.x86_64.rpm
[root]# rpm -i ./oracle-instantclient11.2-odbc-11.2.0.4.0-1.x86_64.rpm
```

3. Create a configuration file in `/etc/ld.so.conf.d` to add the Oracle client libraries to the `LD_LIBRARY_PATH`.

   To confirm where the RPMs installed the libraries, run `rpm -qlp <rpmFileName>`:

```
[root]# cat >/etc/ld.so.conf.d/oracle-instant-client.conf <<EOL
/usr/lib/oracle/11.2/client64/lib
EOL
```

   > If you installed Oracle Instant Client from a repository, run `repoquery -ql <rpmName>` instead.

   Rebuild the `LD_LIBRARY_PATH`:

```
[root]# ldconfig
```

4. Connect to the database using `sqlplus`. If you used RPMs to install the client, the 32-bit and 64-bit clients are already in your PATH.

```
[root]# sqlplus64 moab/moab@adaptive-oracle:1561/MOAB

SQL*Plus: Release 11.2.0.4.0 Production on Mon Sep 30 14:35:10 2013

Copyright (c) 1982, 2013, Oracle.  All rights reserved.

Connected to:
Oracle Database 11g Release 11.2.0.1.0 - 64bit Production
```

   The 64-bit `sqlplus` client was used to connect to a 64-bit 11g instance called MOAB, which is hosted on `adaptive-oracle.ac`.

5. Verify that you are logged in to the correct database:

```
SQL> select name from v$database
  2  ;
NAME
---------
MOAB
```

6. Create the database in Oracle using the `moab-db-oracle-create.sh` script located in the `contrib/sql` directory.

   This contrib/sql directory is in the expanded tarball directory.

   Useful comments are at the top of the script. Read the comments before running the script.

```
[root]# ./moab-db-oracle-create.sh
```

7. Display all of user's `MOAB` tables:

```
SQL> select table_name from all_tables where owner = 'MOAB';

TABLE_NAME
------------------------------
TRIGGERS
MOAB
OBJECTTYPE
VCS
EVENTTYPE
JOBHISTORY
MCHECKPOINT
NODES
EVENTS
NODESTATSGENERICRESOURCES
JOBS
RESERVATIONS
GENERICMETRICS
REQUESTS
```

```
GENERALSTATS
NODESTATS

16 rows selected.
SQL>
```

8. Generate a script to describe all of user's `MOAB` tables. Cut and paste the following into a terminal that is *not* logged in to SQLPlus:

```
[root]# cat > /tmp/generateDescribe.sql <<EOL
SET HEADING OFF
SET FEEDBACK OFF
SET ECHO OFF
SET PAGESIZE 0
SPOOL /tmp/describeAllUserTables.sql
select 'desc '||owner||'.'||table_name||';' from all_tables where owner = 'MOAB';
SPOOL OFF
EOL
```

9. Run `describeAllUserTables.sql`:

```
[root]# SQL> start /tmp/describeAllUserTables.sql
```

Related Topics

- 19.2 Connecting to a MySQL Database with an ODBC Driver
- 19.3 Connecting to a PostgreSQL Database with an ODBC Driver

19.5 Migrating Your Database to Newer Versions of Moab

Sometimes when upgrading from an older version of Moab to a newer version, you must update your database schema. If the schema Moab expects to operate against is different from the actual schema of the database Moab is connected to, Moab might not be able to use the database properly and data might be lost.

When upgrading the Moab database schema from an old version, you must perform each version upgrade in order. **You cannot skip versions.**

In this section:

19.5.1 Migrate from Moab 9.1 to Moab 10.0

In Moab Workload Manager 10.0, some obsolete node attributes were removed (e.g., `ResOvercommitFactor`) from the ODBC database schema. To upgrade your database with these changes, use the `moab-db-<database>-upgrade10_0.sql` file located in the `contrib/sql` directory.

This contrib/sql directory is in the expanded tarball directory.

For example, to migrate your PostgreSQL database from the 9.1 schema to the 10.0 schema, run the following:

```
[postgres]$ psql Moab < moab-db-postgresql-upgrade10_0.sql
```

The database name is usually `Moab`.

Similar migration scripts exist for Oracle and MySQL.

19.5.2 Migrate from Moab 9.0 to Moab 9.1

In Moab Workload Manager 9.1, the former reservation statistics (STATCAPS, STATCIPS, STATTAPS and STATTIPS) were replaced with (STATCBPS, STATCRPS, STATTBPS and STATTRPS) in the ODBC database schema. To upgrade your database with these changes, use the `moab-db-<database>-upgrade9_1.sql` file located in the contrib/sql directory.

This contrib/sql directory is in the expanded tarball directory.

For example, to migrate your PostgreSQL database from the 9.0 schema, run the following:

```
[postgres]$ psql Moab < moab-db-postgresql-upgrade9_1.sql
```

The database name is usually `Moab`.

Similar migration scripts exist for Oracle and MySQL.

19.5.3 Migrate from Moab 8.1 to Moab 9.0

There were no schema changes between Moab 8.1 and Moab 9.0, therefore there is no migration script that needs to be run to adapt your database from Moab 8.1 to Moab 9.0.

19.5.4 Migrate from Moab 8.0 to Moab 8.1

In Moab Workload Manager 8.1, a new accounting event 'AMCONTINUE was added and the datatypes of some reservation statistics were changed. To upgrade your database with these changes, use the moab-db-<database>-upgrade8_1.sql file located in the `contrib/sql` directory.

This contrib/sql directory is in the expanded tarball directory.

For example, to migrate your MySQL database from the 8.0 (or later) schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade8_1.sql
```

The database name is usually `Moab`.

Similar migration scripts exist for Oracle and PostgreSQL.

19.5.5 Migrate from Moab 7.5 to Moab 8.0

In Moab Workload Manager 8.0, column names that have become reserved words in newer versions of MySQL, PostgreSQL, and Oracle were renamed to eliminate the need to quote column names in SQL statements. Also, a few additional columns were added to existing tables to support Moab's Green feature. To upgrade your database with these changes, use the `moab-db-<database>-upgrade8_0.sql` file located in the `contrib/sql` directory.

This contrib/sql directory is in the expanded tarball directory.

For example, to migrate your MySQL database from the 7.5 (or later) schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade8_0.sql
```

The database name is usually `Moab`.

Similar migration scripts exist for Oracle and PostgreSQL.

19.5.6 Migrate from Moab 7.2.6-7.2.10 to Moab 7.5

In Moab Workload Manager 7.5, column names that are reserved words in databases supported by Adaptive Computing were renamed to eliminate the need to quote column names in SQL statements. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_5.sql` file located in the `contrib/sql` directory.

This contrib/sql directory is in the expanded tarball directory.

For example, to migrate your MySQL database from the 7.2.6 (or later) schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade7_5.sql
```

The database name is usually `Moab`.

Similar migration scripts exist for Oracle and PostgreSQL.

19.5.7 Migrate from Moab 7.2.0-7.2.5 to Moab 7.2.6

In Moab Workload Manager 7.2.6, several columns were extended and the primary key on the Triggers table changed. To upgrade your database with these changes, use the `moab-db-<database>-upgrade7_2_6.sql` file located in the `contrib/sql` directory.

This contrib/sql directory is in the expanded tarball directory.

For example, to migrate your MySQL database from the 7.2.x (pre-7.2.6) schema to the 7.2.6 schema, run the following:

```
[root@]# mysql -u root -D <database name> -p < moab-db-mysql-upgrade7_2_6.sql
```

The database name is usually `Moab`.

Similar migration scripts exist for Oracle and PostgreSQL.

The 7.2.6 database upgrade is compatible with all earlier versions of 7.2.

19.6 Importing Statistics from stats/DAY.* to the Moab Database

The `contrib/stat_converter` folder contains the files to build `mstat_converter`, an executable that reads file-based statistics in a Moab stats directory and dumps them into a database. It also reads the Moab checkpoint file (`.moab.ck`) and dumps that to the database also. It uses the `$MOABHOMEDIR/moab.cfg` file to connect to the appropriate database and reads the statistics files from `$MOABHOMEDIR/stats`.

To run, execute the program `mstat_converter` with no arguments.

The statistics converter program does not clear the database before converting. However, if there are statistics in the database and the statistics files from the same period, the converter reports duplicate data errors.

Chapter 20: Accelerators

Moab can integrate with the Torque resource manager to discover, report, schedule, and submit workload to various accelerator architectures (such as NVIDIA GPUs or Intel® Xeon Phi™ coprocessor architecture) for parallel processing. See the topics below for specific information.

In this chapter:

20.1 Scheduling GPUs

In this section:

20.1.1 Deploying and Configuring GPUs

There are several ways you can deploy and configure Moab/Torque systems with GPUs.

Method 1 (recommended): GPUs with NVIDIA/NVML and cgroups

- The only method that guarantees that users do not use more GPUs than they request.
- Ensures that multiple users will not attempt to use the same GPU.
- Torque will attempt to use CPUs and memory that are close to the GPUs to allow the jobs to execute faster and more consistently.
- Provides all of the information on the GPUs that configuring with NVML does (method 2).

- In addition to configuring NVIDIA GPU support (see 20.3 NVIDIA GPUs), enabling cgroups requires installing hwloc libraries. See 'Torque NUMA-Aware Configuration' in the *Torque Resource Manager Administrator Guide* for instructions.

Method 2: GPUs with NVIDIA/NVML, but without cgroups (see 20.3 NVIDIA GPUs)

- Guarantees that you detect an accurate number of GPUs without manual configuration.
- Provides reporting information on the state of each GPU.
- Allows users to set a mode for each GPU in use by their job, as needed.

> When using this method, `pbs_server` automatically appends '`gpus=<count>`' to the end of the line in `TORQUE_HOME/server_priv/nodes` for any node with a GPU, overriding any such manual configuration.

Method 3: Configuring GPUs in the nodes file (see 20.1.2 Using GPUs with Minimal Configuration)

- Allows jobs to request GPUs.
- Requires manual configuration.
- Does not guarantee accuracy.

20.1.2 Using GPUs with Minimal Configuration

You can request GPUs on a node at job submission by specifying a nodes resource request, using the *qsub -l* option. The number of GPUs a node has must be specified in the nodes file. The GPU is then reported in the output of *pbsnodes*:

```
napali
state = free
np = 2
ntype = cluster
status = rectime=1288888871,varattr=,jobs=,state=free,netload=1606207294,gres=tom:!
/home/dbeer/dev/scripts/dynamic_
resc.sh,loadave=0.10,ncpus=2,physmem=3091140kb,availmem=32788032348kb,
totmem=34653576492kb,idletime=4983,nusers=3,nsessions=14,sessions=3136 1805 2380 2428
1161 3174 3184
3191 3209 3228 3272 3333 20560 32371,uname=Linux napali 2.6.32-25-generic #45-Ubuntu
SMP Sat Oct 16 19:52:42
UTC 2025 x86_64,opsys=linux
mom_service_port = 15002
mom_manager_port = 15003
gpus = 1
```

The $PBS_GPUFILE has been created to include GPU awareness. The GPU appears as a separate line in $PBS_GPUFILE and follows this syntax:

```
<hostname>-gpu<index>
```

If a job was submitted to run on a server called 'napali' (the submit command looks something like: *qsub test.sh -l nodes=1:ppn=2:gpus=1*), the $PBS_GPUFILE contains:

```
napali-gpu0
```

It is left up to the job's owner to make sure that the job executes properly on the GPU. By default, Torque treats GPUs exactly the same as ppn (which corresponds to CPUs).

Related Topics

- 20.2 Using GPUs with NUMA
- 20.3 NVIDIA GPUs

20.2 Using GPUs with NUMA

The pbs_server requires awareness of how the MOM is reporting nodes since there is only one MOM daemon and multiple MOM nodes. Configure the server_priv/nodes file with the num_node_boards and numa_gpu_node_str attributes. The attribute num_node_boards tells pbs_server how many NUMA nodes are reported by the MOM. If each NUMA node has the same number of GPUs, add the total number of GPUs to the nodes file. Following is an example of how to configure the nodes file with num_node_boards:

```
numahost gpus=12 num_node_boards=6
```

This line in the nodes file tells pbs_server there is a host named numahost and that it has 12 GPUs and 6 nodes. The pbs_server divides the value of GPUs (12) by the value for num_node_boards (6) and determines there are 2 GPUs per NUMA node.

In this example, the NUMA system is uniform in its configuration of GPUs per node board, but a system does not have to be configured with the same number of GPUs per node board. For systems with non-uniform GPU distributions, use the attribute numa_gpu_node_str to let pbs_server know where GPUs are located in the cluster.

If there are equal numbers of GPUs on each NUMA node, you can specify them with a string. For example, if there are 3 NUMA nodes and the first has 0 GPUs, the second has 3, and the third has 5, you add this to the nodes file entry:

```
numa_gpu_node_str=0,3,5
```

In this configuration, pbs_server knows it has three MOM nodes and the nodes have 0, 3s, and 5 GPUs respectively. Note that the attribute gpus is not used. The gpus attribute is ignored because the number of GPUs per node is specifically given.

`qsub` supports the mapping of `-l gpus=X` to `-l gres=gpus:X`. This enables users who are using NUMA systems to make requests such as `-l ncpus=20,gpus=5` (or `-l ncpus=20:gpus=5`)indicating they are not concerned with the GPUs in relation to the NUMA nodes they request; they only want a total of 20 cores and 5 GPUs.

The `qsub -l gpus=X` option is deprecated. We recommend that you request GPUs using the resource request 2.0 syntax `-L` (see '-L NUMA Resource Request' in the *Torque Resource Manager Administrator Guide*).

Related Topics

- 20.1 Scheduling GPUs
- 20.3 NVIDIA GPUs

20.3 NVIDIA GPUs

In this section:

20.3.1 Using NVIDIA GPUs

This document assumes that you have installed the NVIDIA CUDA ToolKit and the NVIDIA development drivers on a compute node with an NVIDIA GPU. (Both can be downloaded from CUDA Toolkit 12.6 Update 3 Downloads).

Severe scheduling performance problems have been observed in systems with GPUs that do not have persistence mode enabled. We strongly recommend doing this on all GPU nodes. See 20.3.7 Enabling Persistence Mode for more information.

CUDA version 6.0 or later is recommended for Torque 6.0 or later. CUDA version 4.1 is the minimum required.

The recommended method for deploying Moab/Torque systems with GPUs is to include NVIDIA/NVML options and cgroups. See 20.1 Scheduling GPUs.

As of version 2.5.6, `pbs_mom` can query for GPU hardware information and report that status to `pbs_server`, adding a `gpustatus` line in the output for `pbsnodes`.

`qsub` includes options for setting the GPU mode and for resetting GPU ECC error counts.

To generate MOM binaries with GPU support, you must build on a system that has the CUDA libraries for that specific GPU hardware and operating system. Because the CUDA toolkit installer refuses to run on a system without a GPU card (and the server typically lacks that hardware), the usual method of building for GPUs involves putting the source on a node with a GPU and compiling there with NVDIA/NVML options. This requires gcc, libtool, and other build utilities, so first you must follow the 'Install Packages' instructions for the Torque server in the *Moab HPC Suite Installation and Configuration Guide* for your version and OS on that host. Once you've configured and built, you can generate the MOM installer by running `make packages`, as described below. `pbs_server` can communicate with `pbs_mom` binaries configured for GPU support regardless the server's build options.

To configure for NVIDIA GPU support, include these options:

- `--enable-nvidia-gpus`
- `--with-nvml-lib=DIR` (library path for `libnvidia-ml.so`)
- `--with-nvml-include=DIR` (include path for `nvml.h`)

`nvml.h` is only found in the NVIDIA CUDA ToolKit.

Example:

```
./configure --with-debug --enable-nvidia-gpus --with-nvml-lib=/usr/lib64 --with-nvml-
include=/cuda/NVML --with-hwloc-path=/usr/lib64/
```

20.3.2 Package Installation/Upgrade

The package files are self-extracting packages that can be copied and executed on your production nodes to do a new installation, or overlay and upgrade existing installations in the same locations. Example:

```
> make packages
Building ./torque-package-clients-linux-x86_64.sh ...
Building ./torque-package-mom-linux-x86_64.sh ...
Building ./torque-package-server-linux-x86_64.sh ...
Building ./torque-package-gui-linux-x86_64.sh ...
Building ./torque-package-devel-linux-x86_64.sh ...
Done.
$
$ ls -l torque-package-*
-rwxr-xr-x 1 root root 2180510 May 19 15:05 torque-package-clients-linux-x86_64.sh
-rwxr-xr-x 1 root root 4066774 May 19 15:05 torque-package-devel-linux-x86_64.sh
-rwxr-xr-x 1 root root  163505 May 19 15:05 torque-package-doc-linux-x86_64.sh
-rwxr-xr-x 1 root root 4813027 May 19 15:05 torque-package-mom-linux-x86_64.sh
-rwxr-xr-x 1 root root 8168502 May 19 15:05 torque-package-server-linux-x86_64.sh
$
$ ./torque-package-clients-linux-x86_64.sh --install

Installing TORQUE archive...

Done.
$
```

When updating, it is good practice to stop the `pbs_server` and make a backup of the Torque home directory. You will also want to backup the output of `qmgr -c "print server"`. The update will only overwrite the binaries. To do a 'rolling upgrade' and have the MOMs automatically start at a safe point (when `pbs_mom` is between jobs), install the new version and then toggle `enablemomrestart` for the existing MOM processes. Example:

```
# pdsh -w node[01-99] /usr/local/sbin/momctl -q enablemomrestart=1
```

Or

```
momctl -q enablemomrestart=1 -h :ALL
```

Refer to the 'Rolling Upgrade' subsection of the appendix 'Considerations Before Upgrading' in the *Torque Resource Manager Administrator Guide* for more information and suggestions.

If you move GPU cards to different slots, you must restart `pbs_server` in order for Torque to recognize the drivers as the same ones in different locations rather than two new, additional drivers.

For more information, see the topics below.

20.3.3 Torque Configuration

CUDA 6.0 or later is recommended for Torque 6.0 or later. CUDA version 4.1 is the minimum required. Using `nvidia-smi` to configure Torque for NVIDIA GPU support is deprecated.

To use the NVML (NVIDIA Management Library) API instead of nvidia-smi, configure Torque using `--with-nvml-lib=DIR` and `--with-nvml-include=DIR`. These commands specify the location of the `libnvidia-ml` library and the location of the `nvml.h` include file.

```
./configure -with-nvml-lib=/usr/lib --with-nvml-include=/usr/local/cuda/Tools/NVML
server_priv/nodes:
node001  gpus=1
node002  gpus=4
...
pbsnodes  -a
node001
    ...
    gpus = 1
...
```

By default, when Torque is configured with `--enable-nvidia-gpus` the `TORQUE_HOME/nodes` file is automatically updated with the correct GPU count for each MOM node.

20.3.4 GPU Modes for NVIDIA 260.x Driver

0 – Default - Shared mode available for multiple processes

1 – Exclusive - Only one COMPUTE thread is allowed to run on the GPU

2 – Prohibited - No COMPUTE contexts are allowed to run on the GPU

Prohibited mode is not allowed for user jobs.

20.3.5 GPU Modes for NVIDIA 270.x Driver

0 – Default - Shared mode available for multiple processes

1 – Exclusive Thread - Only one COMPUTE thread is allowed to run on the GPU (v260 exclusive)

2 – Prohibited - No COMPUTE contexts are allowed to run on the GPU

Prohibited mode is not allowed for user jobs.

3 – Exclusive Process - Only one COMPUTE process is allowed to run on the GPU

20.3.6 gpu_status

```
root@gpu:~# pbsnodes gpu
gpu
...
    gpus = 2
    gpu_status = gpu[1]=gpu_id=0:6:0;gpu_product_name=Tesla
    C2050;gpu_display=Disabled;gpu_pci_device_id=6D110DE;gpu_pci_location_id=0:6:0;
    gpu_fan_speed=54 %;gpu_memory_total=2687 Mb;gpu_memory_used=74
Mb;gpu_mode=Default;gpu_state=Unallocated;gpu_utilization=96
%;gpu_memory_utilization=10
%;gpu_ecc_mode=Enabled;gpu_single_bit_ecc_errors=0;gpu_double_bit_ecc_errors=
0;gpu_temperature=88 C,gpu[0]=gpu_id=0:5:0;gpu_product_name=Tesla
C2050;gpu_display=Enabled;gpu_pci_device_id=6D110DE;gpu_pci_location_id=0:5:0;
gpu_fan_speed=66 %;gpu_memory_total=2687 Mb;gpu_memory_used=136
Mb;gpu_mode=Default;gpu_state=Unallocated;gpu_utilization=96
%;gpu_memory_utilization=10
%;gpu_ecc_mode=Enabled;gpu_single_bit_ecc_errors=0;
gpu_double_bit_ecc_errors=0;gpu_temperature=86 C,driver_ver=270.41.06,timestamp=Wed
May 4 13:00:35
2024
```

20.3.7 Enabling Persistence Mode

When `pbs_mom` has been built with NVIDIA support and linked to NVML libraries, `pbs_mom` regularly calls `nvmlInit()`. `nvmlInit` can take a long time to complete when persistence mode has not been enabled for NVIDIA devices. We recommend that you enable persistence mode for NVIDIA GPUs. You can enable persistence mode using `nvidia-smi` (as root):

```
nvidia-smi -i <target gpu> -pm ENABLED
```

Or

```
nvidia-smi -i <target gpu> --persistence-mode ENABLED
```

When the `-i` option is not specified, `nvidia-smi` enables persistence mode for all NVIDIA GPUs. The effect of this operation is immediate; however, it does not persist across reboots. After each reboot, persistence mode will default to `DISABLED`. To have persistence mode persist across reboots, run the NVIDIA Persistence Daemon. See the web page NVIDIA GPU Management and Deployment for more information on persistence mode, and the web page Persistence Daemon for more information on the Persistence Daemon.

20.3.8 Requesting GPUs and Setting GPU Mode

If you are using CUDA 8 or newer, the default of `exclusive_thread` is no longer supported. If the server specifies an `exclusive_thread` setting, the MOM will substitute an `exclusive_process` mode setting. We recommend that you set the default to `exclusive_process`.

`qsub` allows specifying required compute mode when requesting GPUs. If no GPU mode is requested, it will default to 'exclusive' for NVIDIA driver version 260 or 'exclusive_thread' for NVIDIA driver version 270 and above.

- `qsub -l nodes=1:ppn=1:gpus=1`
- `qsub -l nodes=1:gpus=1`
- `qsub -l nodes=1:gpus=1:default`
- `qsub -l nodes=1:gpus=1:shared` ('shared' and 'default' are both recognized and are equivalent)
- `qsub -l nodes=1:gpus=1:exclusive_thread`
- `qsub -l nodes=1:gpus=1:exclusive_process`
- `qsub -l nodes=1:gpus=1:reseterr`
- `qsub -l nodes=1:gpus=1:reseterr:exclusive_thread (exclusive_thread:reseterr)`
- `qsub -l nodes=1:gpus=1:reseterr:exclusive_process`

Related Topics

- 20.1 Scheduling GPUs
- 20.2 Using GPUs with NUMA

20.4 GPU Metrics

GPU metrics can be collected for nodes that:

- Have one or more GPUs.
- Use NVIDIA drivers v260.x or v270.x.

GPU metric tracking must be enabled in `moab.cfg`:

```
RMCFG[torque]  flags=RECORDGPUMETRICS
```

There is one GPU metric for all GPU devices within a node (gpu_timestamp) and nine GPU metrics for each GPU device within a node. If the maximum GPU devices within a node is 4, you must increase the MAXGMETRIC value in `moab.cfg` by (maxgpudevices x gpumetrics) + 1. In this case, the formula is (4 x 9) + 1 = 37, so whatever the `MAXGMETRIC` value is, it must be increased by 37. This way, when enabling GPU metrics recording, Moab has enough GMETRIC types to accommodate the GPU metrics.

GPU Metrics Map

The GPU metric names map is as follows (where `X` is the GPU number):

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
timestamp	gpu_timestamp The gpu_timestamp metric is global to all GPUs on the node and indicates the last time the driver collected information on the GPUs.	The time data was collected in epoch time
gpu_fan_ speed	gpuX_fan	The current fan speed as a percentage
gpu_ memory_ total	gpuX_mem	The total GPU memory in megabytes
gpu_ memory_ used	gpuX_usedmem	The total used GPU memory in megabytes
gpu_ utilization	gpuX_util	The GPU capability currently in use as a percentage
gpu_ memory_ utilization	gpuX_memutil	The GPU memory currently in use as a percentage
gpu_ecc_	gpuX_ecc	Whether ECC is enabled

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
mode		or disabled
gpu_single_ bit_ecc_ errors	gpuX_ecc1err	The total number of EEC single-bit errors since the last counter reset
gpu_double_ bit_ecc_ errors	gpuX_ecc2err	The total number of EEC double-bit errors since the last counter reset
gpu_ temperature	gpuX_temp	The GPU current temperature in Celsius

Example 20-1: GPU

```
$ mdiag -n -v --xml

<Data>
<node AGRES="GPUS=2;"
AVLCLASS="[test 8][batch 8]"
CFGCLASS="[test 8][batch 8]"
GMETRIC="gpu1_fan:59.00,gpu1_mem:2687.00,gpu1_usedmem:74.00,gpu1_util:94.00,gpu1_
memutil:9.00,gpu1_ecc:0.00,gpu1_ecc1err:0.00,gpu1_ecc2err:0.00,gpu1_temp:89.00,gpu0_
fan:70.00,gpu0_mem:2687.00,gpu0_usedmem:136.00,gpu0_util:94.00,gpu0_memutil:9.00,gpu0_
ecc:0.00,gpu0_ecc1err:0.00,gpu0_ecc2err:0.00,gpu0_temp:89.00,gpu_
timestamp:1304526680.00"
GRES="GPUS=2;"
LASTUPDATETIME="1304526518" LOAD="1.050000"
MAXJOB="0" MAXJOBPERUSER="0" MAXLOAD="0.000000" NODEID="gpu"
NODEINDEX="0" NODESTATE="Idle" OS="linux" OSLIST="linux"
PARTITION="makai" PRIORITY="0" PROCSPEED="0" RADISK="1"
RAMEM="5978" RAPROC="7" RASWAP="22722" RCDISK="1" RCMEM="5978"
RCPROC="8" RCSWAP="23493" RMACCESSLIST="makai" SPEED="1.000000"
STATMODIFYTIME="1304525679" STATTOTALTIME="315649"
STATUPTIME="315649"></node>
</Data>
```

20.5 Intel® Xeon Phi™ Coprocessor Configuration

In this section:

20.5.1 Intel Many-Integrated Cores (MIC) Architecture Configuration

If you use an Intel Many-Integrated Cores (MIC) architecture-based product (e.g., Intel Xeon Phi™) in your cluster for parallel processing, you must configure Torque to detect them.

In this topic:

20.5.1.A Prerequisites

- Torque 4.2 or later
- If you set up Torque using auto-detection and intend to get the MIC-based device status report, you must build pbs_mom on a system that has the lower-level API libraries for the MIC-based device(s) installed. Additionally, every MOM built with `--enable-mics` and running on a compute node must already have the lower-level API libraries installed on the node. Note that the library is called `coi_host`. You must obtain the API libraries from Intel.

20.5.1.B Setup Options

There are two ways to configure MIC-based devices with Torque: (1) manually and (2) by auto-detection.

Manual Configuration

Add `mics=X` to the nodes file for the appropriate nodes. See 'Specifying Compute Nodes' in the *Torque Resource Manager Administrator Guide* for more information.

```
napali np=12 mics=2
```

Auto-Detect

When you use auto-detection, pbs_mom discovers the MIC-based devices and reports them to pbs_server.

At build time, add `--enable-mics` to the configure line:

```
./configure --enable-mics <other configure options>
```

20.5.2 Validating the Configuration

In this topic:

20.5.2.A Torque

pbsnodes

Example 20-2: pbsnodes output

```
slesmic
       state = free
       np = 100
       ntype = cluster
       status =
rectime=1347634381,varattr=,jobs=,state=free,netload=7442004852,gres=,loadave=0.00,ncp
us=32,physmem=65925692kb,availmem=66531344kb,totmem=68028984kb,idletime=59059,nusers=2
,nsessions=8,sessions=4387 4391 4392 4436 4439 4443 4459 100395,uname=Linux slesmic
3.0.13-0.27-default #1 SMP Wed June 15 13:33:49 UTC 2024 (d73692b) x86_64,opsys=linux
       mom_service_port = 15002
       mom_manager_port = 15003
       mics = 2
       mic_status = mic[1]=mic_id=8796;num_cores=61;num_threads=244;physmem=8065748992;
free_physmem=7854972928;swap=0;free_swap=0;max_frequency=1090;isa=COI_ISA_
KNC;load=0.000000;normalized_load=0.000000;,mic[0]=mic_id=8796;num_cores=61;num_
threads=244;physmem=8065748992;free_physmem=7872712704;swap=0;free_swap=0;max_
frequency=1090;isa=COI_ISA_KNC;load=0.540000;normalized_load=0.008852;

rhmic.ac
       state = free
       np = 100
       ntype = cluster
       status =
rectime=1347634381,varattr=,jobs=,state=free,netload=3006171583,gres=,loadave=0.00,ncp
us=32,physmem=65918268kb,availmem=66901588kb,totmem=67982644kb,idletime=59477,nusers=2
,nsessions=2,sessions=3401 29320,uname=Linux rhmic.ac 2.6.32-220.el6.x86_64 #1 SMP Tue
Dec 6 19:48:22 GMT 2024 x86_64,opsys=linux
       mom_service_port = 15002
       mom_manager_port = 15003
       mics = 1
       mic_status = mic[0]=mic_id=8796;num_cores=61;num_threads=244;physmem=8065748992;
free_physmem=7872032768;swap=0;free_swap=0;max_frequency=1090;isa=COI_ISA_
KNC;load=0.540000;normalized_load=0.008852;<mic_status>;
```

20.5.2.B Moab

mdiag -n -v

Example 20-3: mdiag -n -v output

```
$ mdiag -n -v
compute node summary
Name                    State   Procs      Memory         Disk          Swap
Speed   Opsys   Arch Par   Load Classes                        Features

hola                     Idle    4:4      8002:8002        1:1       10236:13723
1.00   linux       - hol   0.24 [batch]                         -
GRES=MICS:2,
-----                     ---    4:4      8002:8002        1:1       10236:13723

Total Nodes: 1  (Active: 0  Idle: 1  Down: 0)
```

checknode -v

Example 20-4: checknode output

```
$ checknode slesmic
node slesmic

State:      Idle  (in current state for 00:00:16)
Configured Resources: PROCS: 100  MEM: 62G  SWAP: 64G  DISK: 1M  MICS: 2
Utilized   Resources: SWAP: 1581M
Dedicated  Resources: ---
Generic Metrics:    mic1_mic_id=8796.00,mic1_num_cores=61.00,mic1_num_
threads=244.00,mic1_physmem=8065748992.00,mic1_free_physmem=7854972928.00,mic1_
swap=0.00,mic1_free_swap=0.00,mic1_max_frequency=1090.00,mic1_load=0.12,mic1_
normalized_load=0.00,mic0_mic_id=8796.00,mic0_num_cores=61.00,mic0_num_
threads=244.00,mic0_physmem=8065748992.00,mic0_free_physmem=7872679936.00,mic0_
swap=0.00,mic0_free_swap=0.00,mic0_max_frequency=1090.00
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.000
Classes:    [batch]
RM[napali]* TYPE=PBS
EffNodeAccessPolicy: SHARED

Total Time: 3:45:43  Up: 3:45:43 (100.00%)  Active: 00:00:00 (0.00%)

Reservations:
  ---
```

20.5.3 Job Submission

In this topic:

20.5.3.A Syntax

Example 20-5: Request MIC-based device(s) in qsub

```
qsub .... -l nodes=X:mics=Y
```

20.5.3.B qstat -f

Example 20-6: qstat -f output

```
Job Id: 5271.napali
Job_Name = STDIN
Job_Owner = dbeer@napali
job_state = Q
queue = batch
server = napali
Checkpoint = u
ctime = Fri Oct 14 08:56:33 2024
Error_Path = napali:/home/dbeer/dev/private-torque/trunk/STDIN.e5271
Hold_Types = n
Join_Path = oe
Keep_Files = n
Mail_Points = a
mtime = Fri Oct 14 08:56:33 2024
Output_Path = napali:/home/dbeer/dev/private-torque/trunk/STDIN.o5271
Priority = 0
qtime = Fri Oct 14 08:56:33 2024
Rerunable = True
Resource_List.neednodes = 1:mics=1
Resource_List.nodect = 1
Resource_List.nodes = 1:mics=1
substate = 10
Variable_List = PBS_O_QUEUE=batch,PBS_O_HOME=/home/dbeer,
       PBS_O_LOGNAME=dbeer,
       PBS_O_PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
       in:/usr/games,PBS_O_MAIL=/var/mail/dbeer,PBS_O_SHELL=/bin/bash,
       PBS_O_LANG=en_US.UTF-8,
       PBS_O_SUBMIT_FILTER=/usr/local/sbin/torque_submitfilter,
       PBS_O_WORKDIR=/home/dbeer/dev/private-torque/trunk,PBS_O_HOST=napali,
       PBS_O_SERVER=napali
euser = dbeer
egroup = company
queue_rank = 3
queue_type = E
etime = Fri Oct 14 08:56:33 2024
submit_args = -l nodes=1:mics=1
fault_tolerant = False
job_radix = 0
```

```
submit_host = napali
```

20.5.3.C checkjob -v

Example 20-7: checkjob -v output

```
dthompson@mahalo:~/dev/moab-test/trunk$ checkjob -v 2
job 2 (RM job '2.mahalo')

AName: STDIN
State: Idle
Creds:  user:dthompson  group:dthompson  class:batch
WallTime:   00:00:00 of 1:00:00
SubmitTime: Thu Sep 13 17:06:06
(Time Queued  Total: 00:00:24  Eligible: 00:00:02)

TemplateSets:  DEFAULT
Total Requested Tasks: 1

Req[0]  TaskCount: 1  Partition: ALL
Dedicated Resources Per Task: PROCS: 1  MICS: 1

...
```

20.6 Intel® Xeon Phi™ Coprocessor Metrics

Intel Many-Integrated Cores (MIC) architecture-based device (e.g., Intel Xeon Phi™) metrics can be collected for nodes that:

- Have one or more MIC-based devices.
- Run Torque 4.2.x or later.
- Run Moab 7.2 or later.

MIC-based device metric tracking must be enabled in `moab.cfg`:

```
RMCFG[torque]  flags=RECORDMICMETRICS
```

There are 11 metrics for each MIC-based device within a node. If the maximum MIC-based devices within a node is 4, you must increase the MAXGMETRIC value in `moab.cfg` by (`maxmicdevices` x `micmetrics`). In this case, the formula is (4 x 11) = 44, so whatever the `MAXGMETRIC` value is, it must be increased by 44. This way, when enabling MIC-based device metrics recording, Moab has enough GMETRIC types to accommodate the additional metrics.

MIC-Based Metrics Map

The MIC-based metric names map is as follows (where X is the MIC-based device number):

Metric name as returned by pbsnodes	GMETRIC name as stored in Moab	Metric output
mic_id	micX_mic_id	The ID of the MIC-based device
num_cores	micX_num_cores	The number of cores in the MIC-based device
num_threads	micX_num_threads	The number of hardware threads on the MIC-based device
physmem	micX_physmem	The total physical memory in the MIC-based device
free_physmem	micX_free_physmem	The available physical memory in the MIC-based device
swap	micX_swap	The total swap space on the MIC-based device
free_swap	micX_free_swap	The unused swap space on the MIC-based device
max_frequency	micX_max_frequency	The maximum frequency speed of any core in the MIC-based device
isa	micX_isa	The hardware interface type of the MIC-based device
load	micX_load	The total current load of the MIC-based device
normalized_load	micX_normalized_load	The normalized load of the MIC-based device (total load divided by number of cores in the MIC-based device)

Chapter 21: Preemption

Sites possess workloads of varying importance, and users might want to run jobs with higher priorities before jobs with lower priorities. This can be done by using preemption. Preemption is simply the process by which a higher-priority job can take the place of a lower-priority job. You can also use preemption for optimistic scheduling and development job support.

This section explains how to configure and use preemption. 21.2.4 Simple Example of Preemption offers a basic introduction and contains examples to help you get started using preemption. The other sections offer more explanation and information about what you can do with preemption and contain some best practices that will help you avoid the need for troubleshooting in the future.

While this section does not explain every possible preemption configuration, it does prescribe the best practices for setting up and using preemption with your system. We recommend that you follow the established instructions contained in this section.

Preemption does not work with dynamic provisioning.

Neither `SPANEVENLY` nor `DELAY` values of the NODESETPLUS parameter will work with multi-req jobs or preemption.

Do not allow preemption with interactive jobs unless `PREEMPTPOLICY` is set to `CANCEL`. For more information, see 21.1.1 Canceling Jobs with Preemption.

In this chapter:

Related Topics

- Chapter 7: Optimizing Scheduling Behavior – Backfill and Node Sets

21.1 Preemption Tasks

In this section:

21.1.1 Canceling Jobs with Preemption

CANCEL is one of the PREEMPTPOLICY types (for more information, see 21.2.3 PREEMPTPOLICY Types). The CANCEL attribute cancels active jobs, regardless of any JOBFLAGS (such as REQUEUEABLE or SUSPENDABLE).

For information about PREEPMPTEE and PREEMPTOR flags, see 21.2.2 Preemption Flags.

You should not allow preemption with interactive jobs unless PREEMPTPOLICY is set to CANCEL.

The following outlines some benefits of using CANCEL and also lists some things you should be aware of if you choose to use it:

- Advantages - This attribute is the easiest to configure and use.
- Cautions - Canceled jobs are not automatically restarted or requeued. Users must resubmit canceled jobs.

To Preempt Jobs Using CANCEL

1. Make the following configurations to the moab.cfg file:
   a. Set the parameter GUARANTEEDPREEMPTION to TRUE. This causes Moab to lock PREEMPTOR jobs until JOBRETRYTIME expires.
   b. Set the parameter PREEMPTIONALGORITHM to specify how Moab handles preemption scheduling policies.

> If you use parameter JOBNODEMATCHPOLICY `EXACTNODE`, you must also add PREEMPTIONALGORITHM PREEMPTORCENTRIC in order for preemption to function reliably.

c. Set `PREEMPTPOLICY` to `CANCEL` (for more information, see 21.2.3 PREEMPTPOLICY Types).

d. Make sure that the `PREEMPTEE` job has a lower priority than the `PREEMPTOR` job (for more information, see 21.2.2 Preemption Flags). For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY CANCEL

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (`test1`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

(Optional) Examine the following output for *showq*:

```
Moab.7
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Running    128       00:01:59      Thu Nov 10 12:28:44

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test2
```

(Optional) Examine the following output for *showq*:

```
Moab.8
[john@g06]# showq

active jobs------------------------
```

```
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Canceling  128       00:01:56      Thu Nov 10 12:28:44
Moab.8    john        Running    128       00:02:00      Thu Nov 10 12:28:48

2 active jobs 128 of 128 processors in use by local jobs (100.00%)
              2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that `test1` is canceled when `test2` is submitted.

(Optional) Examine the *`checkjob`* outputs for these two jobs:

```
[john@g06]$ checkjob Moab.9
job Moab.9

State: Removed
Completion Code: -1 Time: Thu Nov 10 12:28:48
Creds: user:john group:john qos:test1
WallTime: 00:00:02 of 00:02:00
SubmitTime: Thu Nov 10 12:28:44
(Time Queued Total: 00:00:07 Eligible: 00:00:00)

Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.UFe8sQ

StartCount: 1
Flags: GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
```

Note that the preempted job has been removed:

```
[john@g06]$ checkjob Moab.10
job Moab.10

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:00:00 of 00:02:00
SubmitTime: Thu Nov 10 12:36:31
```

```
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Thu Nov 10 12:28:48
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.CZavjU

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.10' (-00:00:07 -> 00:01:53 Duration: 00:02:00)
```

Related Topics

- 21.2.2 Preemption Flags
- 21.2.3 PREEMPTPOLICY Types
- 21.2.5 Testing and Troubleshooting Preemption

21.1.2 Checkpointing Jobs with Preemption

`CHECKPOINT` is one of the PREEMPTPOLICY types (for more information, see 21.2.3 PREEMPTPOLICY Types). For systems that allow checkpointing, the `CHECKPOINT` attribute enables a job to save its current state and either terminate or continue running. A checkpointed job can restart at any time and resume execution from its most recent checkpoint.

You can tune checkpointing behavior on a per-resource manager-basis by setting the `CHECKPOINTSIG` and `CHECKPOINTTIMEOUT` attributes of the RMCFG parameter.

For information about `PREEPMPTEE` and `PREEMPTOR` flags, see 21.1 Preemption Tasks.

The following outlines some benefits of using `CHECKPOINT` and also lists some things you should be aware of if you choose to use it:

- Advantages - This attribute enables you to restart a job from its last checkpoint.
- Cautions - Jobs tend to take longer to complete when you use `CHECKPOINT`.

To Preempt Jobs Using CHECKPOINT

Make the following configurations to the `moab.cfg` file:

1. Set the parameter GUARANTEEDPREEMPTION to TRUE. This causes Moab to lock PREEMPTOR jobs until JOBRETRYTIME expires. This locks the job on a node and keeps trying to preempt.
2. Set the parameter PREEMPTIONALGORITHM to specify how Moab handles preemption scheduling policies.

   If you use parameter JOBNODEMATCHPOLICY EXACTNODE, you must also add PREEMPTIONALGORITHM PREEMPTORCENTRIC in order for preemption to function reliably.

3. Set PREEMPTPOLICY to CHECKPOINT (for more information, see 21.2.3 PREEMPTPOLICY Types).
4. Make sure that the PREEMPTEE job has a lower priority than the PREEMPTOR job (for more information, see 21.2.2 Preemption Flags).

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY CHECKPOINT

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

Related Topics

- 21.2.2 Preemption Flags
- 21.2.3 PREEMPTPOLICY Types
- 21.2.5 Testing and Troubleshooting Preemption

21.1.3 Requeuing Jobs with Preemption

REQUEUE is one of the PREEMPTPOLICY types (for more information, see 21.2.3 PREEMPTPOLICY Types). The REQUEUE value terminates active jobs and returns them to the job queue in an idle state.

For information about PREEPMPTEE and PREEMPTOR flags, see 21.1 Preemption Tasks.

The following outlines some benefits of using REQUEUE and also lists some things you should be aware of if you choose to use it:

- Advantages - Jobs are automatically resubmitted into the job queue.

- Cautions - A job gets resubmitted in the job queue at the same priority it had when Moab originally started it (i.e., the job does not jump ahead in the queue). Jobs start over from the beginning.

You must mark a job as `RESTARTABLE` if you want it to requeue. If you do not, the job will be canceled when it is preempted.

If supported by the resource manager, you can set the `RESTARTABLE` job flag when submitting the job by using the msub -r option. Otherwise, use the JOBFLAGS attribute of the associated class or QoS credential, as in this example:

```
CLASSCFG[low] JOBFLAGS=RESTARTABLE
```

To Preempt Jobs Using REQUEUE

1. Make the following configurations to the `moab.cfg` file:
   a. Set the parameter GUARANTEEDPREEMPTION to `TRUE`. This causes Moab to lock `PREEMPTOR` jobs until JOBRETRYTIME expires.
   b. Set the parameter PREEMPTIONALGORITHM to specify how Moab handles preemption scheduling policies.

      If you use parameter JOBNODEMATCHPOLICY `EXACTNODE`, you must also add PREEMPTIONALGORITHM PREEMPTORCENTRIC in order for preemption to function reliably.

   c. Set `PREEMPTPOLICY` to `REQUEUE` (for more information, see 21.2.3 PREEMPTPOLICY Types).
   d. Make sure that the `PREEMPTEE` job has a lower priority than the `PREEMPTOR` job (for more information, see 21.2.2 Preemption Flags). For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY REQUEUE

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (`test1`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

(Optional) Examine the following output for *showq*:

```
Moab.1
[john@g06]# showq
```

```
active jobs------------------------
JOBID     USERNAME    STATE     PROCS     REMAINING      STARTTIME
Moab.1    john        Running   128       00:09:59       Wed Nov 9 15:56:33

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test2 -l procs=128
```

(Optional) Examine the following output for *`showq`* and *`checkjob`*:

```
Moab.2
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME     STATE     PROCS    REMAINING     STARTTIME
Moab.2    john         Running   128      00:09:59      Wed Nov 9 15:56:47

1 active job 128 of 128 processors in use by local jobs (100.00%)
             2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE       PROCS    WCLIMIT       QUEUETIME
Moab.1    john        Idle        128      00:10:00      Wed Nov 9 15:56:33

1 eligible job

blocked jobs-----------------------
JOBID     USERNAME    STATE       PROCS    WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

```
[john@g06]# checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:02:04 of 00:10:00
SubmitTime: Wed Nov 9 15:56:46
(Time Queued Total: 00:00:01 Eligible: 00:00:00)

StartTime: Wed Nov 9 15:56:47
Total Requested Tasks: 128
```

```
Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.ELoX5Q

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.2' (-00:02:21 -> 00:07:39 Duration: 00:10:00)
```

Related Topics

- 21.2.2 Preemption Flags
- 21.2.3 PREEMPTPOLICY Types
- 21.2.5 Testing and Troubleshooting Preemption

21.1.4 Suspending Jobs with Preemption

`SUSPEND` is one of the PREEMPTPOLICY types (for more information, see 21.2.3 PREEMPTPOLICY Types). The `SUSPEND` attribute causes active jobs to stop executing, but to remain in memory on the allocated compute nodes.

For information about `PREEPMPTEE` and `PREEMPTOR` flags, see 21.1 Preemption Tasks.

The following outlines some benefits of using `SUSPEND` and also lists some things you should be aware of if you choose to use it:

- Advantages - The job remains in memory on the allocated compute nodes. Using `SUSPEND` frees up processor resources. The job can restart where it left off before it was suspended.
- Cautions - There is a possibility that having multiple suspended jobs on a compute node will crash the swap. Moab tracks only *requested* memory of active jobs (not *used* memory). The swap can crash if the job uses a lot of memory and Moab starts other jobs. Suspended jobs do not relinquish their licenses.

You must mark a job as SUSPENDABLE if you want it to suspend. If you do not, the job will be requeued or canceled when it is preempted.

If supported by the resource manager, you can set the job SUSPENDABLE flag when submitting the job by using the msub -r option. Otherwise, use the JOBFLAGS attribute of the associated class or QoS credential, as in this example:

```
CLASSCFG[low] JOBFLAGS=SUSPENDABLE
```

To Preempt Jobs Using SUSPEND

When you use SUSPEND, you must increase your JOBRETRYTIME. By default, JOBRETRYTIME is set to 60 seconds, but when you use SUSPEND, we recommend that you increase the time to 300 seconds (5 minutes).

1. Make the following configurations to the moab.cfg file:
   a. Set the parameter GUARANTEEDPREEMPTION to TRUE. This causes Moab to lock PREEMPTOR jobs until JOBRETRYTIME expires.
   b. Set the parameter PREEMPTIONALGORITHM to specify how Moab handles preemption scheduling policies.

      If you use parameter JOBNODEMATCHPOLICY EXACTNODE, you must also add PREEMPTIONALGORITHM PREEMPTORCENTRIC in order for preemption to function reliably.

   c. Set PREEMPTPOLICY to SUSPEND (for more information, see 21.2.3 PREEMPTPOLICY Types).
   d. For the PREEMPTEE job, set JOBFLAGS=RESTARTABLE,SUSPENDABLE.
   e. Make sure that the PREEMPTEE job has a lower priority than the PREEMPTOR job (for more information, see 21.2.2 Preemption Flags). For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY SUSPEND

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE,SUSPENDABLE MEMBERULIST=john
PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to the preemptee QoS (test1). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test1
```

   (Optional) Examine the output for *showq*:

```
Moab.7
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Running    128       00:01:59      Thu Nov 10 12:28:44

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

3. Now submit a job to the preemptor QoS (`test2`). For example:

```
[john@g06]$ echo sleep 120 | msub -l procs=128,walltime=120 -l qos=test2
```

(Optional) Examine the output for *`showq`*:

```
Moab.8
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.7    john        Suspended  128       00:01:56      Thu Nov 10 12:28:44
Moab.8    john        Running    128       00:02:00      Thu Nov 10 12:28:48

2 active jobs 128 of 128 processors in use by local jobs (100.00%)
              2 of 2 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that when a job is suspended, it stays in the output of *`showq`*. This is normal behavior for a suspended job. Moab should only suspend a job once.

(Optional) Examine the *`checkjob`* outputs for these two jobs:

```
[john@g06]$ checkjob Moab.9
job Moab.9
```

```
State: Suspended
Creds: user:john group:john qos:test1
WallTime: 00:00:02 of 00:02:00
SubmitTime: Thu Nov 10 12:36:29
(Time Queued Total: 00:00:07 Eligible: 00:00:00)

Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.UFe8sQ

StartCount: 1
Flags: RESTARTABLE,SUSPENDABLE,PREEMPTEE,GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
job cannot be resumed: preemption required but job is conditional preemptor with no
targets
BLOCK MSG: non-idle state 'Running' (recorded at last scheduling iteration)
```

```
[john@g06]$ checkjob Moab.10
job Moab.10

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:00:00 of 00:02:00
SubmitTime: Thu Nov 10 12:36:31
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Thu Nov 10 12:36:31
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64


IWD: /opt/native
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.CZavjU

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.10' (-00:00:07 -> 00:01:53 Duration: 00:02:00)
```

Occasionally, Moab will keep a job from restarting, holding it in a suspended state for a long period of time, if it thinks the job cannot restart. For example, if a job could write to I/O before it was suspended, and now it cannot, Moab would realize the job is unable to start and leave it in a suspended state.

Related Topics

- 21.2.2 Preemption Flags
- 21.2.3 PREEMPTPOLICY Types
- 21.2.5 Testing and Troubleshooting Preemption

21.1.5 Using Owner Preemption

Owner preemption enables jobs submitted by a reservation owner to preempt jobs submitted by other users (for more information, see 6.1.5 Configuring and Managing Reservations).

Owner preemption is enabled with the OWNERPREEMPT reservation flag.

For information about `PREEPMPTEE` and `PREEMPTOR` flags, see 21.2.2 Preemption Flags.

To Enable Owner Preemption

1. Make the following configurations to the `moab.cfg` file:
    a. Set the parameter GUARANTEEDPREEMPTION to `TRUE`. This causes Moab to lock `PREEMPTOR` jobs until JOBRETRYTIME expires.
    b. Make sure that the parameter JOBNODEMATCHPOLICY is *not* set to `EXACTNODE`, which is not currently supported for preemption (for more information, see 21.2.5 Testing and Troubleshooting Preemption).
    c. Set the `PREEMPTPOLICY` type (for more information, see 21.2.3 PREEMPTPOLICY Types).
    d. Set the `OWNERPREEMPT` flag.

    Optionally, if you want the owner preemption to override any PREEMPTMINTIME settings for `PREEMPTEE` jobs, you can set the OWNERPREEMPTIGNOREMINTIME flag also.

    e. Specify an owner.

    If the non-owner job does not have a `RESTARTABLE` or `REQUEUEABLE` flag set, the job will cancel.

    For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY <policy>

SRCFG[myrez] FLAGS=OWNERPREEMPT HOSTLIST=node01
SRCFG[myrez] OWNER=USER:john
SRCFG[myrez] USERLIST=jane,john PERIOD=INFINITY

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=restartable MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=10000
```

2. Submit a job to a user who is not the owner (in this example, `jane`):

```
[jane@g06]$ echo sleep 600 | msub -l walltime=600 -l procs=64
```

(Optional) Examine the following output for *`showq`* and *`checkjob`* for `jane`'s job:

```
Moab.1
[jane@g06]$ showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS      REMAINING     STARTTIME
Moab.1    jane        Running    64         00:09:57      Mon Nov 14 12:07:52

1 active job     64 of 64 processors in use by local jobs (100.00%)
                 1 of 1 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

```
root@g06]# checkjob Moab.1
job Moab.1

State: Running
Creds: user:jane group:jane
WallTime: 00:01:02 of 00:10:00
SubmitTime: Mon Nov 14 12:07:52
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Mon Nov 14 12:07:52
Total Requested Tasks: 64

Req[0] TaskCount: 64 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:64]


IWD: /opt/native
SubmitDir: /opt/native
```

```
Executable: /opt/native/spool/moab.job.FoZfIU

StartCount: 1
Flags: GLOBALQUEUE,PROCSPECIFIED
StartPriority: 1
Reservation 'Moab.1' (-00:01:24 -> 00:08:36 Duration: 00:10:00)
```

3. Now submit a job for the owner (in this example, `john`):

```
[john@g06]$ echo sleep 600 | msub -l walltime=600 -l procs=50
```

(Optional) Examine the following output for *`showq`* and *`checkjob`* for `john`'s job:

```
Moab.2
[john@g06]$ showq

active jobs------------------------
JOBID     USERNAME    STATE      PROCS     REMAINING     STARTTIME
Moab.1    jane        Canceling  64        00:07:43      Mon Nov 14 12:07:52
Moab.2    john        Running    50        00:09:59      Mon Nov 14 12:10:08

2 active jobs    64 of 64 processors in use by local jobs (100.00%)
                 1 of 1 nodes active (100.00%)

eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that `jane`'s job is canceled once `john`'s job is submitted:

```
[john@g06]$ checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john
WallTime: 00:00:31 of 00:10:00
SubmitTime: Mon Nov 14 12:10:08
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Mon Nov 14 12:10:08
Total Requested Tasks: 50

Req[0] TaskCount: 50 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:50]


IWD: /opt/native
```

```
SubmitDir: /opt/native
Executable: /opt/native/spool/moab.job.jf1N4a

StartCount: 1
Flags: HASPREEMPTED,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 1
Reservation 'Moab.2' (-00:00:48 -> 00:09:12 Duration: 00:10:00)
```

Note the new **HASPREEMPTED** flag.

(Optional) Now look at the *showq* for `jane`'s job (after):

```
[root@g06]# checkjob Moab.1
job Moab.1

State: Removed
Completion Code: -1 Time: Mon Nov 14 12:10:08
Creds: user:jane group:jane
WallTime: 00:02:47 of 00:10:00
SubmitTime: Mon Nov 14 12:07:52
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

Total Requested Tasks: 64

Req[0] TaskCount: 64 Partition: FLEXlm
NodeCount: 1

Allocated Nodes:
[node01:64]


IWD: /opt/native
Executable: /opt/native/spool/moab.job.FoZfIU

Execution Partition: FLEXlm
Flags: GLOBALQUEUE,PROCSPECIFIED
StartPriority: 0
```

Note that the state is now **Removed**.

Related Topics

- 21.2.2 Preemption Flags
- 21.2.3 PREEMPTPOLICY Types
- 21.2.5 Testing and Troubleshooting Preemption

21.1.6 Using QoS Preemption

This section describes how to configure the `moab.cfg` file to set up preemption with QoS. Using QoS, you can specify preemption rules and control access to preemption privileges by using the QFLAGS `PREEMPTEE` and `PREEMPTOR` credentials. For information about the `PREEMPTEE` and `PREEMPTOR` flags, see 21.2.2 Preemption Flags.

QoS-based preemption only occurs when the following three conditions are satisfied:

- The preemptor job has the `PREEMPTOR` attribute set.
- The preemptee job has the `PREEMPTEE` attribute set.
- The preemptor job has a higher priority than the preemptee job.

To Configure moab.cfg for QoS Preemption

1. Set the parameter GUARANTEEDPREEMPTION to `TRUE`. This causes Moab to lock `PREEMPTOR` jobs until JOBRETRYTIME expires.
2. Make sure that the parameter JOBNODEMATCHPOLICY is *not* set to `EXACTNODE`, which is not currently supported for preemption (for more information, see 21.2.5 Testing and Troubleshooting Preemption).
3. If it is not already, set the parameter NODEACCESSPOLICY to `SHARED`.
4. Set the PREEMPTPOLICY policy type (for more information, see 21.2.3 PREEMPTPOLICY Types).
5. Set up `QFLAGS` to mark jobs as `PREEMPTEE` (a lower-priority job that can be preempted by a higher-priority job), or as `PREEMPTOR` (a higher-priority job that can preempt a lower-priority job), as in the example:

   ```
   QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
   QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
   ```

   For more information, see 21.2.2 Preemption Flags.
6. Make sure that the `PREEMPTEE` job has a lower priority than the `PREEMPTOR` job, as in the example:

   ```
   QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
   QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
   ```

For example:

```
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY <policy>

QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000
```

Related Topics

- 21.2.2 Preemption Flags
- 21.2.3 PREEMPTPOLICY Types

21.2 Preemption Reference

In this section:

21.2.1 Manual Preemption Commands

You can use the mjobctl command to manually preempt jobs. The command can modify a job's execution state in the following ways:

Action	Flag	Details
Cancel	-c	Terminate job; remove from queue
Checkpoint	-C	Terminate and checkpoint job leaving job in queue
Requeue	-R	Terminate job; leave in queue
Resume	-r	Resume suspended job
Start (execute)	-x	Start idle job
Suspend	-s	Suspend active job

In general, users are allowed to suspend or terminate jobs they own. Admins are allowed to suspend, terminate, resume, and execute any queued jobs.

21.2.2 Preemption Flags

Using QoS, you can specify preemption rules and control access to preemption privileges. This enables you to increase system throughput, improve job response time for specific classes of jobs, or enable various political policies. You enable all policies by specifying some QoS credentials with the QFLAGS `PREEMPTEE`, and others with `PREEMPTOR`.

PREEMPTEE	
Description	Indicates that the job can be preempted by a higher-priority job.
Use	Use for lower-priority jobs that can be preempted.
Notes	This might delay some node actions. When reprovisioning, the system job might expire before the provision action occurs; while the action will still occur, the job will not show it.
Example	`QOSCFG[test1] QFLAGS=PREEMPTEE MEMBERULIST=<user> PRIORITY=100`

PREEMPTOR	
Description	Indicates that the job should take priority and preempt any `PREEMPTEE` jobs.
Use	Use for jobs that need to take precedence over lower-priority jobs.
Notes	`PREEMPTOR` jobs, either queued or running, must have a higher priority than `PREEMPTEE` jobs. When you configure a job as a `PREEMPTOR`, you should also increase its priority (for details, see the parameters PREEMPTPRIOJOBSELECTWEIGHT and PREEMPTRTIMEWEIGHT).
Example	`QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=<user> PRIORITY=10000`

Additional Preemptor and Preemptee Information

Preemptor priority plays a big role in preemption. Generally, you should assign the preemptor job a higher priority value than any other queued jobs so that it will move to (or near to) the top of the eligible queue.

You can set the RESERVATIONPOLICY parameter to `NEVER`. With this configuration, preemptee jobs can start whenever idle resources become available. These jobs will be allowed to run until a preemptor job arrives, at which point the preemptee jobs are preempted, freeing the resource. This configuration allows near immediate resource access for the preemptor jobs. Using this approach, a cluster can maintain near 100% system utilization while still delivering excellent turnaround time to the most important jobs.

In environments where job checkpointing or job suspension incur significant overhead, you might want to constrain the rate at which job preemption is allowed. You can use the JOBPREEMPTMINACTIVETIME parameter to throttle job preemption. In essence, this parameter prevents a newly started or newly resumed job from being eligible for preemption until it has executed for a specified amount of time. Conversely, you can exclude jobs from preemption after they have run for a certain amount of time by using the JOBPREEMPTMAXACTIVETIME parameter.

Related Topics

- 21.1.6 Using QoS Preemption

21.2.3 PREEMPTPOLICY Types

You can use the PREEMPTPOLICY parameter to control how the scheduler preempts a job. This parameter enforces preemption using one of the following methods:

PREEMPTPOLICY Type	Description
SUSPEND	Causes active jobs to stop executing, but to remain in memory on the allocated compute nodes.
CHECKPOINT	Saves the current job state and either terminates or continues running the job. A checkpointed job can restart at any time and resume execution from its most recent checkpoint.
REQUEUE	Terminates active jobs and returns them to the job queue in an idle state.
CANCEL	Cancels active jobs.

Each of these methods varies in the level of disruption to the job, `SUSPEND` being the least disruptive and `CANCEL` being the most disruptive.

Moab uses preemption escalation to free up resources. So for example, if the `PREEMPTPOLICY` is set to `SUSPEND`, Moab uses this method if it is available; however, Moab will escalate it to something potentially more disruptive if necessary to preempt and free up resources.

Related Topics

- 21.1.1 Canceling Jobs with Preemption
- 21.1.2 Checkpointing Jobs with Preemption
- 21.1.3 Requeuing Jobs with Preemption
- 21.1.4 Suspending Jobs with Preemption

21.2.4 Simple Example of Preemption

This topic illustrates the process of setting up preemption on your system from beginning to end and contains examples of what actions to take and what you should see as you proceed.

In this topic:

21.2.4.A Scenario

Example Scenario

For this basic setup example, we will have a user who can submit to either a 'test1' or 'test2' QoS. This example will use a `REQUEUE` preemption type.

We will go through three parts to set up this preemption:

- Configuring the `moab.cfg` file
- Submitting a job to the `PREEMPTEE` QoS
- Submitting a job to the `PREEMPTOR` QoS

21.2.4.B Configuring moab.cfg

First, you will need to make some configurations to the `moab.cfg` file.

1. Set the parameter GUARANTEEDPREEMPTION to TRUE (this causes Moab to lock PREEMPTOR jobs until JOBRETRYTIME expires).
2. Make sure that the parameter JOBNODEMATCHPOLICY is *not* set to EXACTNODE, which is not currently supported for preemption (for more information, see 21.2.5 Testing and Troubleshooting Preemption.
3. Set the PREEMPTPOLICY type. In this example, PREEMPTPOLICY is set to REQUEUE. For more information, see 21.2.3 PREEMPTPOLICY Types.
4. Set up QFLAGS to mark jobs as PREEMPTEE (a lower-priority job that can be preempted by a higher-priority job), or as PREEMPTOR (a higher-priority job that can preempt a lower-priority job). For more information, see 21.2.2 Preemption Flags.

   For this example, we also set JOBFLAGS=RESTARTABLE (because this example uses REQUEUE). For more information, see 21.1.3 Requeuing Jobs with Preemption.

5. Make sure that the PREEMPTEE job has a lower priority than the PREEMPTOR job.

Here is an example of how that all looks in a moab.cfg file (text marked **bold** for emphasis):

```
GUARANTEEDPREEMPTION TRUE
#should not be JOBNODEMATCHPOLICY EXACTNODE as it causes problems when starting jobs

PREEMPTPOLICY REQUEUE

QOSCFG[test1] QFLAGS=PREEMPTEE JOBFLAGS=RESTARTABLE MEMBERULIST=john PRIORITY=100
QOSCFG[test2] QFLAGS=PREEMPTOR MEMBERULIST=john PRIORITY=1000
```

Now you can submit a job to the preemptee QoS (test1).

21.2.4.C Submitting a Job to the Preemptee QoS

Let's submit a job to the preemptee QoS (test1), requesting all processor cores in the cluster:

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test1 -l procs=128
```

Take a look at the *showq* and *checkjob* output:

```
Moab.1
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME     STATE      PROCS      REMAINING      STARTTIME
Moab.1    john         Running    128        00:09:59       Wed Nov 9 15:56:33

1 active job     128 of 128 processors in use by local jobs (100.00%)
                 2 of 2 nodes active (100.00%)
```

```
eligible jobs----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 eligible jobs

blocked jobs-----------------------
JOBID     USERNAME    STATE     PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total job: 1
```

```
[john@g06]# checkjob Moab.1
job Moab.1

State: Running
Creds: user:john group:john qos:test1
WallTime: 00:00:00 of 00:10:00
SubmitTime: Wed Nov 9 15:56:33
(Time Queued Total: 00:00:00 Eligible: 00:00:00)

StartTime: Wed Nov 9 15:56:33
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses

Allocated Nodes:
node[01-02]*64

IWD: /opt/native/
SubmitDir: /opt/native/
Executable: /opt/native/spool/moab.job.zOyf1N

StartCount: 1
Flags: RESTARTABLE,PREEMPTEE,GLOBALQUEUE,PROCSPECIFIED
Attr: PREEMPTEE
StartPriority: 100
Reservation 'Moab.1' (-00:00:03 -> 00:09:57 Duration: 00:10:00
```

21.2.4.D Submitting a Job to the Preemptor QoS

Now we will submit a preemptor QoS job (`test2`) to preempt the first job (`test1`):

```
[john@g06]# echo sleep 600 | msub -l walltime=600 -l qos=test2 -l procs=128
```

Examine the following output for *`showq`* and *`checkjob`*:

```
Moab.2
[john@g06]# showq

active jobs------------------------
JOBID     USERNAME    STATE     PROCS     REMAINING     STARTTIME
Moab.2    john        Running   128       00:09:59      Wed Nov 9 15:56:47

1 active job 128 of 128 processors in use by local jobs (100.00%)
             2 of 2 nodes active (100.00%)
```

```
eligible jobs----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME
Moab.1    john        Idle       128        00:10:00      Wed Nov  9 15:56:33

1 eligible job

blocked jobs-----------------------
JOBID     USERNAME    STATE      PROCS      WCLIMIT       QUEUETIME

0 blocked jobs

Total jobs: 2
```

Note that the preemptor job (Moab.2) moved to **Running**, while the preemptee job (Moab.1) was requeued.

Note the flag **HASPREEMPTED**; this is set when the `PREEMPTOR` job has preempted the `PREEMPTEE` job:

```
[john@g06]# checkjob Moab.2
job Moab.2

State: Running
Creds: user:john group:john qos:test2
WallTime: 00:02:04 of 00:10:00
SubmitTime: Wed Nov  9 15:56:46
(Time Queued Total: 00:00:01 Eligible: 00:00:00)

StartTime: Wed Nov  9 15:56:47
Total Requested Tasks: 128

Req[0] TaskCount: 128 Partition: licenses
NodeCount: 2

Allocated Nodes:
node[01-02]*64


IWD: /opt/native/
SubmitDir: /opt/native/
Executable: /opt/native/spool/moab.job.ELoX5Q

StartCount: 1
Flags: HASPREEMPTED,PREEMPTOR,GLOBALQUEUE,PROCSPECIFIED
StartPriority: 10000
Reservation 'Moab.2' (-00:02:21 -> 00:07:39 Duration: 00:10:00)
```

Also note that the preemptor job priority plays a very big role in preemption. Generally, you should assign the preemptor a higher priority than any other queued jobs so that it will move to (or near to) the top of the eligible queue.

21.2.5 Testing and Troubleshooting Preemption

There are multiple steps associated with setting up a working preemption policy. With preemption, issues arise because it appears that Moab is not allowing preemptor jobs to preempt preemptee jobs in the right way. To diagnose this, use the following checklist:

	Verify that preemptor jobs are marked with the `PREEMPTOR` flag. Verify with checkjob <JOBID> | grep Flags.
	Verify that preemptee jobs are marked with the `PREEMPTEE` flag. Verify with checkjob <JOBID> | grep Flags.
	Verify that the start priority of the preemptor job is higher than the priority of the preemptee job. Verify with checkjob <JOBID> | grep Priority.
	Verify that the resources allocated to the preemptee job match those requested by the preemptor job.
	Verify that there are no policies preventing preemption from occurring. Verify with checkjob -v -n <NODEID> <JOBID>.
	Verify that the parameter PREEMPTPOLICY is properly set. See 21.2.3 PREEMPTPOLICY Types.
	Verify that the preemptee job is properly marked as restartable, suspendable, or checkpointable. Verify with checkjob <JOBID> | grep Flags.
	Verify that the parameter GUARANTEEDPREEMPTION is set to `TRUE`.
	If the parameter JOBNODEMATCHPOLICY is set to `EXACTNODE`, verify PREEMPTIONALGORITHM is set to `PREEMPTORCENTRIC`.
	Verify that the parameter NODEACCESSPOLICY is *not* set to `SINGLEUSER`. `SHARED` is recommended.
	Verify that the parameter BACKFILLPOLICY is set to `FIRSTFIT`.
	Verify that the resource manager is properly responding to preemption requests. (Use mdiag -R.)
	If there is a resource manager level race condition, verify that Moab is properly holding target resources. Verify with mdiag -S and set the parameter RESERVATIONRETRYTIME if needed.

Related Topics

- 6.3.3 Managing QoS Access
- 9.4 Checkpoint/Restart Facilities

- 17.3.7 Trigger Components
- Appendix A: Moab Parameters - JOBMAXPREEMPTPERITERATION
- Appendix A: Moab Parameters - ENABLEFSVIOLATIONPREEMPTION
- Appendix A: Moab Parameters - PREEMPTPRIOJOBSELECTWEIGHT
- Appendix A: Moab Parameters - PREEMPTSEARCHDEPTH
- Appendix A: Moab Parameters - USAGEEXECUTIONTIMEWEIGHT (control priority of suspended jobs)
- Appendix A: Moab Parameters - IGNOREPREEMPTEEPRIORITY (relative job priority is ignored in preemption decisions)
- Appendix A: Moab Parameters - DISABLESAMECREDPREEMPTION (jobs cannot preempt other jobs with the same credential)
- Appendix A: Moab Parameters - PREEMPTRTIMEWEIGHT (add remaining time of jobs to preemption calculation)

Chapter 22: Job Templates

A Moab job template is a set of preconfigured settings, attributes, and resources that Moab applies to jobs that match certain criteria or to which you manually apply it. They perform three primary functions:

1. They generically match and categorize jobs.
2. They set arbitrary default or forced attributes for certain jobs.
3. They generate workflows that create and maintain user-requested services in a cloud environment.

You can use job templates in many aspects of scheduling, including Peer-Based Grid usage policies. Job templates are defined using the JOBCFG configuration parameter.

Two methods exist for applying job templates to jobs. You can use the JOBMATCHCFG parameter to mark a template that contains the criteria a job must meet for eligibility and another template as the one to be applied to the job if it is eligible. This enables you to automate the use of templates. For example, to force all interactive jobs to run on a certain set of nodes, you can set one template (the criteria template) to have the `interactive` flag, then give the other template the desired host list. You can also apply a template directly to a job at submission if that ability is enabled for that template.

In this chapter:

22.1 Job Template Tasks

In this section:

22.1.1 Creating Job Templates

Job templates are created in the Moab configure file using the JOBCFG parameter.

To Create a Job Template

1. Open `moab.cfg`. Add the `JOBCFG` parameter and give the new job template a unique name:

   ```
   JOBCFG[newtemplate]
   ```

2. Configure any desired attributes (see 22.2.1 Job Template Extension Attributes). Some of the important attributes include:

   - FLAGS - Lets you specify any job flags that should be applied:

     ```
     JOBCFG[newtemplate] FLAGS=SUSPENDABLE
     ```

     When Moab applies `newtemplate` to a job, the job is marked as suspendable.

   - SELECT - Lets you apply the template directly at job submission:

     ```
     JOBCFG[newtemplate] FLAGS=SUSPENDABLE SELECT=TRUE
     ```

     When you submit a job via *`msub`*, you can specify that your job has `newtemplate` applied to it. When Moab applies the template to a job, that job is marked as suspendable.

   - TEMPLATEDEPEND - Lets you create dependencies when you create a job template workflow (see 22.1.5 Creating Workflows with Job Templates):

     ```
     JOBCFG[newtemplate] FLAGS=SUSPENDABLE SELECT=TRUE TEMPLATEDEPEND=AFTER:job1.pre
     ```

     When Moab applies `newtemplate` to a job, the job cannot run until job `job1.pre` has finished running; the job is also marked as suspendable. You can specify that Moab apply this template to a job as you submit it.

3. If you want to automate job template application, see 22.1.3 Applying Templates Based on Job Attributes for instructions. If you want to apply the template manually on job submission, see 22.1.4 Requesting Job Templates Directly for instructions.

Related Topics

- 22.2.1 Job Template Extension Attributes
- 22.2.3 Job Template Examples

22.1.2 Viewing Job Templates

To View a Job Template

Run the *mdiag -j* command with the policy flag:

```
> mdiag -j --flags=policy --blocking
```

Moab returns a list of job templates configured in moab.cfg.

22.1.3 Applying Templates Based on Job Attributes

The JOBMATCHCFG parameter enables you to establish relationships between a number of job templates. JMAX and JMIN function as filters to determine whether a job is eligible for a subsequent template to be applied to the job. If a job is eligible, JDEF and JSET templates apply attributes to the job. See 22.2.1 Job Template Extension Attributes for more information about the JOBMATCHCFG attributes. The table on that page indicates which job template types are compatible with which job template extension attributes.

JSETs and JDEFs have only been tested using msub as the job submission command.

To Apply a Job Template Based on Job Attributes

1. In the Moab configuration file, create a job template with a set of criteria that a job must meet in order for Moab to apply the template. In the following example, Moab will apply a template to all interactive jobs, so the first template sets the interactive flag:

   ```
   JOBCFG[inter.min] FLAGS=interactive
   ```

2. Create the job template that Moab should apply to the job if it meets the requirements set in the first template. In this example, Moab ignores all configured policies, so the second template sets the ignpolicies flag:

   ```
   JOBCFG[inter.set] FLAGS=ignpolicies
   ```

3. Use the JOBMATCHCFG parameter and its JMAX or JMIN (specify the template specifying maximum or minimum requirements) and JDEF or JSET (specify the template to be applied) attributes to demonstrate the relationship between the two templates (see 22.2.2 Job Template Matching Attributes for more information). In this case, all interactive jobs ignore policies; in other words, if a submitted job has at least the inter.min template settings, Moab applies the inter.set template settings to the job.

```
JOBMATCHCFG[interactive] JMIN=inter.min JSET=inter.set
```

Moab applies the `inter.set` template to all jobs with the `interactive` flag set, causing them to ignore Moab's configured policies.

4. To control which job template is applied to a job that matches multiple templates, use `FLAGS=BREAK`. Job templates are processed in the order they are listed in the configuration file and using the `BREAK` flag causes Moab to stop evaluating `JOBMATCHCFG` entries that occur after the current match.

```
JOBMATCHCFG[small] JMIN=small.min JMAX=small.max JSET.set=small.set FLAGS=BREAK
JOBMATCHCFG[large] JMIN=large.min JMAX=large.max JSET=large.set
```

In this case, the large template is not applied when a job matches both the small and large templates. The small template matches first, and because of `FLAGS=BREAK`, Moab stops evaluating further `JOBMATCHFG` entries for the job.

Related Topics

- 22.2.3 Job Template Examples

22.1.4 Requesting Job Templates Directly

When a job template has its SELECT attribute set to `TRUE`, you can request that template directly on job submission.

To Directly Request Job Templates

1. Set the `SELECT` attribute on the template in `moab.cfg`:

```
JOBCFG[medium.set] NODESET=ONEOF:FEATURE:fast,slow SELECT=true
```

2. Submit a job with a resource list (msub -l), requesting the template using the format `template=<templateName>`:

```
> msub -l template=medium.set
```

Moab creates a job with the `medium.set` job template created in step 1.

Attributes set in the template are evaluated as if they were part of the job submission. They are still subject to all of the same ACLs and policies.

22.1.5 Creating Workflows with Job Templates

Moab can create workflows from individual jobs using job templates.

To Build a Workflow with Job Templates

1. Create the jobs in the workflow using the JOBCFG parameter (see 22.1.1 Creating Job Templates for more information). It might be useful to add the PURGEONSUCCESSONLY flag to your setup or destroy jobs; it will allow you to restart the jobs easily if they fail. Specify the order in which they should run with the TEMPLATEDEPEND attribute. See 9.5.2 Job Dependency Syntax for a list of dependency options.

```
JOBCFG[setup.pre]   TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/setup.pre.sh
JOBCFG[setup.pre2]  TEMPLATEDEPEND=AFTER:setup.pre SELECT=TRUE
EXEC=/nfs/tools/setup.pre2.sh
JOBCFG[engineering] TEMPLATEDEPEND=AFTER:setup.pre2
```

When Moab applies the `engineering` template to a qualifying job, the job will not run until template job `setup.pre` and then `setup.pre2` are created from the specified `EXEC` strings and finish running.

> The Moab naming convention for jobs created with job templates is `<moabId>.<templateName>`. By default, when Moab submits jobs to only one resource manager, the job IDs are synchronized with the resource manager's job IDs. You can use the parameter USEMOABJOBID so that a template-created job is easily associated with its parent job (such as `moab.1`, `moab.1.setup.pre`).

2. Create the job template that will act as the criteria a job must meet for Moab to apply the `engineering` template. In this situation, the job must be submitted with the account name `engineering`:

```
JOBCFG[engineering.match] ACCOUNT=engineering
```

3. Create the JOBMATCHCFG configuration to tell Moab that when a job matches the `engineering.match` template, it should apply the `engineering` template:

```
JOBMATCHCFG[engineering.job] JMIN=engineering.match JSET=engineering
```

Related Topics

- 22.2.1 Job Template Extension Attributes
- 22.2.4 Job Template Workflow Examples

22.2 Job Template Reference

In this section:

22.2.1 Job Template Extension Attributes

When creating a job template, you can use any attribute acceptable within the WIKI workload query data format. In addition, job templates can use any of the extension attributes in the following table. Note that the Template Type (`JMIN`, `JMAX`, `JDEF`, `JSET`) row indicates compatibility with the associated attribute (see 22.1.3 Applying Templates Based on Job Attributes for more information).

Attributes set in a template are evaluated as if they were part of the original job submission. Their jobs are still subject to all the same ACLs and policies.

Attributes

ACCOUNT	GNAME	PRIORITY	TASKS
CLASS	GRES	PRIORITYF	TASKPERNODE
CPUCLOCK	GROUP	QOS	TEMPLATEDEPEND
CPULIMIT	MEM	RARCH	UNAME
DESCRIPTION	NODEACCESSPOLICY	RFEATURES	USER
DPROCS	NODES	RM	VARIABLE
ENV	NODESET	ROPSYS	WCLIMIT
EXEC	PARTITION	SELECT	
FLAGS	PREF	SYSTEMJOBTYPE	

ACCOUNT	
Format	<ACCOUNT>[,<ACCOUNT>]...
Template	`JMIN`

ACCOUNT	
Type	`JDEF` `JSET`
Description	Account credentials associated with job. This is used for job template matching.

Example	`JOBCFG[public] FLAGS=preemptee`
	`JOBCFG[public.min] ACCOUNT=public_acct`
	`JOBMATCHCFG[public] JMIN=public.min JSET=public`

CLASS	
Format	<CLASS>[,<CLASS>]...

Template Type	`JMIN`
	`JDEF`
	`JSET`

Description	Class credentials associated with job. This is used for job template matching.

Example	`JOBCFG[night] FLAGS=preemptor`
	`JOBCFG[night.min] CLASS=night_class`
	`JOBMATCHCFG[night] JMIN=night.min JSET=night`

CPUCLOCK	
Format	<STRING>

Template Type	`JMIN`
	`JMAX`
	`JSET`

Description	CPU clock frequency for all CPUs of a job. For more information, see the job extension CPUCLOCK. The job template extension overrides the job script.

Example	`JOBCFG[slow] SELECT=TRUE cpuclock=1400`
	`JOBCFG[fast] SELECT=TRUE cpuclock=3200`
	
	`JOBCFG[cpu.min] CPUCLOCK=1000`
	`JOBCFG[cpu.max] CPUCLOCK=2000`
	`JOBCFG[cpu.set] CPUCLOCK=1500`
	
	`JOBMATCHCFG[cpu] JMIN=cpu.min JMAX=cpu.max JSET=cpu.set`

CPULIMIT	
Format	[[[DD:]HH:]MM:]SS
Template Type	JMIN JMAX JDEF JSET
Description	Maximum amount of CPU time used by all processes in the job.

Example	`JOBCFG[job.min] CPULIMIT=1:00:00:00`
	`JOBCFG[job.max] CPULIMIT=2:00:00:00`

DESCRIPTION	
Format	<STRING>
Template Type	JMAX JDEF
Description	Description of the job. When you run the checkjob command, the description appears as Description.
Example	`JOBCFG[webdb] DESCRIPTION="Template job"`

DPROCS	
Format	<INTEGER>
Template Type	JMIN JMAX JSET
Description	Number of processors dedicated per task. The default is 1.

Example	`JOBCFG[job.min] DPROCS=2`
	`JOBCFG[job.max] DPROCS=4`

ENV	
Format	<STRING>
Template Type	`JSET`
Description	Adds the specified job environment variables to the job.
Example	`JOBCFG[container] ENV=PBS_CONTAINERINFO=centos`

EXEC	
Format	<STRING>
Template Type	`JSET`
Description	Specifies what the job runs, regardless of what the user set.
Example	`JOBCFG[setup.pre] EXEC=nfs/tools/setup.pre.sh`

FLAGS	
Format	<JOBFLAG>[,<JOBFLAG>]...

Template Type	`JADMIN`
	`JDEF`
	`JSET`

Description	One or more valid job flag values.
Example	`JOBCFG[webdb] FLAGS=NORMSTART`

GNAME	
Format	<STRING>
Template Type	`JDEF` `JSET`

GNAME	
Description	Group credential associated with job.
Example	`JOBCFG[webserv] GNAME=service` For matching the group, see the GROUP attribute.

GRES	
Format	<genericResource>[:<COUNT>][,<genericResource>[:<COUNT>]]...

Template Type	`JMIN`
	`JMAX`
	`JDEF`

Description	Consumable generic attributes associated with individual nodes or the special pseudo-node global, which provides shared cluster (floating) consumable resources. Use the NODECFG parameter to configure such resources.
Example	`JOBCFG[gres.set] GRES=abaqus:2` *In this example, the `gres.set` template applies two Abaqus licenses per task to a matched job.*

GROUP	
Format	<GROUP>[,<GROUP>]...
Template Type	`JMIN`
Description	Group credentials associated with job. This is used for job template matching.
Example	`JOBCFG[webserv] GROUP=service` For information about setting the group, see the GNAME attribute.

MEM	
Format	<INTEGER>

Template Type	`JMIN`
	`JMAX`
	`JDEF`
	`JSET`

Description	Maximum amount of physical memory per task used by the job in megabytes. You can optionally specify other units with your integer (`300kb` or `2gb`, for example). See 'Requesting Resources' in the *Torque Resource Manager Administrator Guide* for more information.
Example	`JOBCFG[smalljobs] MEM=25`

NODEACCESSPOLICY	
Format	One of the following: SHARED, SHAREDONLY, SINGLEJOB, SINGLETASK, SINGLEUSER, or UNIQUEUSER
Template Type	`JDEF` `JSET`
Description	Specifies how node resources will be shared by a job. See 4.3 Node Access Policies for more information.
Example	`JOBCFG[serverapp] NODEACCESSPOLICY=SINGLEJOB`

NODES	
Format	<INTEGER>

Template Type	`JMIN`
	`JMAX`
	`JSET`

Description	Number of nodes required by the job. The default is 1. See 2.3.2 Nodes for more information. When using `JSET`: • If the taskcount of the job is *less* than the NODES value, Moab will modify the taskcount to match the NODES value.

NODES	
	• If the taskcount of the job is *greater* than the NODES value, Moab will attempt to evenly divide the tasks. If the taskcount is not evenly divisible by the NODES value, the job is rejected.

Example	`JOBCFG[job.min] NODES=2`
	`JOBCFG[job.max] NODES=4`

NODESET	
Format	<STRING>
Template Type	JSET
Description	See 7.3 Node Sets for more information.

Example	`JOBCFG[medium.set]`
	`NODESET=ONEOF:FEATURE:fast,slow`

PARTITION	
Format	<PARTITION>[:<PARTITION>]...
Template Type	JMIN JDEF JSET
Description	The partition (or partitions) where a job must run.
Example	`JOBCFG[meis] PARTITION=math:geology`

PREF	
Format	<FEATURE>[,<FEATURE>]...
Template Type	JDEF JSET
Description	Specifies which node features are preferred by the job and should be allocated if available. See the job extension PREF for more information.

PREF	
Example	`JOBCFG[meis] PREF=bigmem`

PRIORITY	
Format	<INTEGER>
Template Type	JMAX JDEF
Description	System job priority. PRIORITY works only as a default setting and not as an override (JSET) setting.
Example	`JOBCFG[meis] PRIORITY=25000`

PRIORITYF	
Format	PRIORITYF='<VALUE>'
Template Type	JSET
Description	Applicable only when using NODEALLOCATIONPOLICY with the PRIORITY format. Lets you change the priority function used to allocate nodes for the job. See the algorithm Node Allocation Policies for available PRIORITY values.
Example	`JOBCFG[limit.set] PRIORITYF='NODEINDEX'`

QOS	
Format	<QOS>[,<QOS>]...
Template Type	JMIN JDEF JSET
Description	QoS credentials associated with job. This is used for job template matching.

QOS	

Example	`JOBCFG[admin] RFEATURES=bigmem`
	`JOBCFG[admin.min] QOS=admin_qos`
	`JOBMATCHCFG[admin] JMIN=admin.min JSET=admin`

RARCH	
Format	<STRING>
Template Type	`JSET`
Description	Architecture required by job.
Example	`JOBCFG[servapp] RARCH=i386`

RFEATURES	
Format	<FEATURE>[,<FEATURE>]...

Template Type	`JMIN`
	`JDEF`
	`JSET`

Description	List of features required by job.

Example	`JOBCFG[servapp]`
	`RFEATURES=fast,bigmem`

RM	
Format	<STRING>
Template Type	`JDEF` `JSET`
Description	Destination resource manager to be associated with job.
Example	`JOBCFG[webdb] RM=torque`

ROPSYS	
Format	<STRING>
Template Type	JDEF JSET
Description	Operating system required by job.
Example	`JOBCFG[test.set] ROPSYS=windows`

SELECT	
Format	<BOOLEAN> : TRUE | FALSE
Description	Job template can be directly requested by job at submission.
Example	`JOBCFG[servapp] SELECT=TRUE`

SYSTEMJOBTYPE	
Template Type	JMIN
Description	System job type (example: vmcreate).

Example	`JOBCFG[vmcreate.min] SYSTEMJOBTYPE=vmcreate`
	`JOBCFG[vmcreate.set]`
	`TRIGGER=atype=reserve,action="00:05:00",etype=end`
	`JOBMATCHCFG[vmcreate] JMIN=vmcreate.min JSET=vmcreate.set`

TASKS	
Format	<INTEGER>
Template Type	JMIN JMAX JSET
Description	Number of tasks required by job. The default is 1. See the component Task Definition for more information.

TASKS	

Example	`JOBCFG[job.min] TASKS=4`
	`JOBCFG[job.max] TASKS=8`

TASKPERNODE	
Format	<INTEGER>

Template Type	`JMIN`
	`JMAX`
	`JDEF`

Description	Exact number of tasks required per node. The default is 0. `TASKPERNODE` works only as a default setting and not as an override (`JSET`) setting.

Example	`JOBCFG[job.min] TASKPERNODE=2`
	`JOBCFG[job.max] TASKPERNODE=4`

TEMPLATEDEPEND	
Format	<TYPE>:<TEMPLATE_NAME>
Description	Create another job from the `<TEMPLATE_NAME>` job template, on which any jobs using this template will depend. This is used for dynamically creating workflows. See 9.5 Job Dependencies for more information. SYNCWITH only supports one dependency with Torque as the resource manager.

Example	`JOBCFG[engineering] TEMPLATEDEPEND=AFTER:setup.pre`
	`JOBCFG[setup.pre] SELECT=TRUE EXEC=/tools/setup.pre.sh`

UNAME	
Format	<STRING>
Default	`JDEF` `JSET`

UNAME	
Description	User credential associated with job.
Example	`JOBCFG[webserv] UNAME=service` For matching the user, see the USER attribute.

USER	
Format	<USER>[,<USER>]...
Template Type	JMIN JMAX
Description	User credentials associated with job.
Example	`JOBCFG[webserv] USER=service` For setting the user, see the UNAME attribute.

VARIABLE	
Format	<NAME>[:<VAL>]
Template Type	JMIN JSET
Description	Variables attached to the job template.
Example	`JOBCFG[this] VARIABLE=var1:1 VARIABLE=var2:1` Variables are set upon successful completion of the job.

WCLIMIT	
Format	[[HH:]MM:]SS
Template Type	JMIN JMAX

WCLIMIT	
	JDEF JSET
Description	Walltime required by job. The default is 8640000 (100 days).

Example	`JOBCFG[job.min] WCLIMIT=2:00:00`
	`JOBCFG[job.max] WCLIMIT=12:00:00`

Related Topics

- 22.1.1 Creating Job Templates

22.2.2 Job Template Matching Attributes

The JOBMATCHCFG parameter enables you to establish relationships between a number of job templates. The table in 22.2.1 Job Template Extension Attributes indicates which job template types are compatible with which job template extension attributes. The following types of templates can be specified with the `JOBMATCHCFG` parameter:

Attribute	Description
JMAX	A potential job is rejected if it has matching attributes set or has resource requests that exceed those specified in this template. For `JMAX`, a job template can specify only positive non-zero numbers as maximum limits for generic resources. If a job requests a generic resource that is not limited by the template, then the template can still be used.
JMIN	A potential job is rejected if it does not have matching attributes set or has resource requests that do not meet or exceed those specified in this template.
JDEF	A matching job has the specified attributes set as defaults but all values can be overridden by the user if the matching attribute is explicitly set at job submission time.
JSET	A matching job has the specified attributes forced to these values and these values override any values specified by the submitter at job submission time.
JSTAT	A matching job has its usage statistics reported into this template.

Related Topics

- 22.1.3 Applying Templates Based on Job Attributes

22.2.3 Job Template Examples

Job templates can be used for a wide range of purposes including enabling automated learning, setting up custom application environments, imposing special account constraints, and applying group default settings. The following examples highlight some of these uses:

Example 22-1: Setting up Application-Specific Environments

```
JOBCFG[xxx] EXEC=*app* JOBPROLOG=/usr/local/appprolog.x
```

Example 22-2: Applying job preferences and defaults

```
JOBCFG[xxx] CLASS=appq EXEC=*app* PREF=clearspeed
NODEALLOCATIONPOLICY PRIORITY
NODECFG[DEFAULT] PRIORITYF=5.0*PREF
```

Example 22-3: Applying Resource Constraints to Fuzzy Collections

In the following example, a job template match is set up. Using the JOBMATCHCFG parameter, Moab is configured to apply all attributes of the `inter.set` job template to all jobs that match the constraints of the `inter.min` job template. In this example, all interactive jobs are assigned the `ignpolicies` flag that allows them to ignore active, idle, system, and partition level policies. Interactive jobs are also locked into the test standing reservation and therefore only allowed to run on the associated nodes.

```
# limit all users to a total of two non-interactive jobs
USERCFG[DEFAULT] MAXJOB=2
SRCFG[test] DESCRIPTION="compute pool for interactive and short duration jobs"
SRCFG[test] JOBATTRLIST=INTERACTIVE
SRCFG[test] MAXTIME=1:00:00
SRCFG[test] HOSTLIST=R:atl[16-63]
JOBCFG[inter.min] FLAGS=interactive
JOBCFG[inter.set] FLAGS=ignpolicies
JOBMATCHCFG[interactive] JMIN=inter.min JSET=inter.set
```

Example 22-4: Resource Manager Templates

In the following example, interactive jobs are not allowed to enter through this resource manager and any job that does route in from this resource manager interface has the `preemptee` flag set:

```
JOBCFG[no_inter] FLAGS=interactive
JOBCFG[preempt_job] FLAGS=preemptee
RMCFG[gridA.in] MAX.JOB=no_inter SET.JOB=preempt_job
```

Related Topics

- 22.1.1 Creating Job Templates

22.2.4 Job Template Workflow Examples

Example 22-5: A Workflow with Multiple Dependencies

In this example, the job will depend on the completion of two other jobs Moab creates:

```
# Engineering2
JOBCFG[engineering2] TEMPLATEDEPEND=AFTER:engineering2.pre2
TEMPLATEDEPEND=AFTER:engineering2.pre
JOBCFG[engineering2.pre2] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/engineering2.pre2.sh
JOBCFG[engineering2.pre] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/engineering2.pre.sh
JOBCFG[engineering2.match] ACCOUNT=engineering2
JOBMATCHCFG[engineering2.job] JMIN=engineering2.match JSET=engineering2
```

Both jobs execute at the same time.

Example 22-6: Jobs that Run After the Submission Job

Three additional jobs are created that depend on the submitted job:

```
# Workflow 2
JOBCFG[workflow2] TEMPLATEDEPEND=BEFORE:workflow2.post1
TEMPLATEDEPEND=BEFORE:workflow2.post2 TEMPLATEDEPEND=BEFORE:workflow2.post3
JOBCFG[workflow2.post1] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post1.sh
JOBCFG[workflow2.post2] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post2.sh
JOBCFG[workflow2.post3] TASKS=2 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow2.post3.sh
JOBCFG[workflow2.match] ACCOUNT=workflow2
JOBMATCHCFG[workflow2.job] JMIN=workflow2.match JSET=workflow2
```

Example 22-7: A Complex Workflow

A complex workflow that handles failures:

```
# Workflow 4
JOBCFG[workflow4.step1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step1.sh TEMPLATEDEPEND=BEFOREFAIL:workflow4.fail1
JOBCFG[workflow4.fail1] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.fail.1.sh TEMPLATEDEPEND=BEFOREANY:workflow4.fail2
JOBCFG[workflow4.fail2] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.fail.2.sh
# Submission job
JOBCFG[workflow4.step2] TEMPLATEDEPEND=AFTEROK:workflow4.step1
TEMPLATEDEPEND=BEFOREOK:workflow4.step3.1 TEMPLATEDEPEND=BEFOREOK:workflow4.step3.2
JOBCFG[workflow4.step3.1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step3.1.sh
```

```
JOBCFG[workflow4.step3.2] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step3.2.sh TEMPLATEDEPEND=BEFOREOK:workflow4.step4
JOBCFG[workflow4.step4] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step4.sh
JOBCFG[workflow4.step4] TEMPLATEDEPEND=BEFOREOK:workflow4.step5.1
TEMPLATEDEPEND=BEFOREOK:workflow4.step5.2 TEMPLATEDEPEND=BEFORENOTOK:workflow4.step5.3
JOBCFG[workflow4.step5.1] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.1.sh
JOBCFG[workflow4.step5.2] TASKS=1 WCLIMIT=00:01:00 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.2.sh
JOBCFG[workflow4.step5.3] TASKS=1 WCLIMIT=00:00:30 SELECT=TRUE
EXEC=/usr/tools/workflow.step5.3.sh
JOBCFG[workflow4.match] ACCOUNT=workflow4
```

Related Topics

- 22.1.3 Applying Templates Based on Job Attributes
- 22.1.5 Creating Workflows with Job Templates

Chapter 23: Moab Workload Manager for Grids

Moab Grid Scheduler enables sites to establish relationships among multiple clusters. There are three types of relationships you can implement within the grid: (1) centralized management, (2) hierarchical management, and (3) localized management. These relationships provide access to additional resources, improve load-balancing, provide single system images, and offer other benefits. The grid interface is flexible allowing sites to establish the needed relationship.

In this chapter:

23.1 Grid Basics

In this section:

23.1.1 Grid Overview

A grid enables you to exchange workload and resource status information and to distribute jobs and data among clusters in an established relationship. In addition, you can use resource reservations to mask reported resources, coordinate requests for consumable resources, and quality of service guarantees.

In a grid, some servers running Moab are a source for jobs (that is, where users, portals, and other systems submit jobs), while other servers running Moab are a destination for these jobs (that is, where the jobs execute). Therefore, jobs originate from a source server and move to a destination server. For a source server to make an intelligent decision, though, resource availability information must flow from a destination server to that source server.

Because you can manage workload on both the source and destination side of a grid relationship, you have a high degree of control over exactly when, how, and where to execute workload.

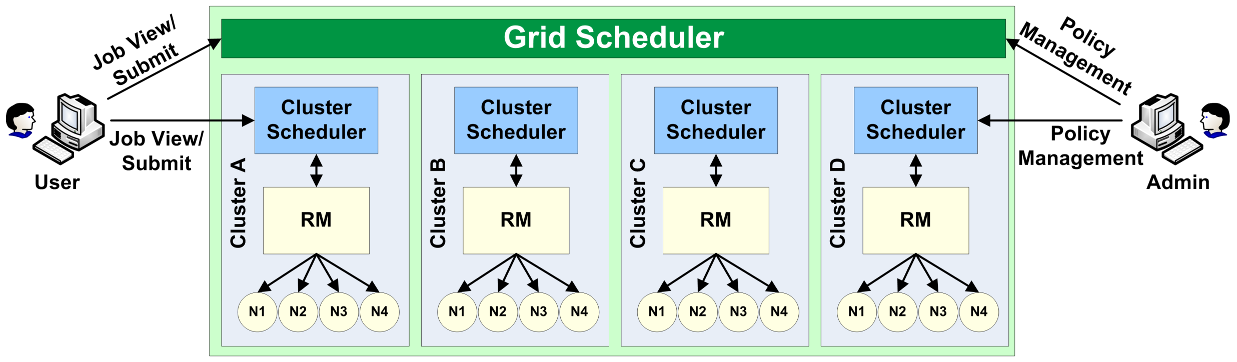
23.1.2 Grid Benefits

Moab's peer-to-peer capabilities can be used for multiple purposes, including any of the following:

- manage access to external shared resources
- enable cluster monitoring information services
- enable massive-scalability clusters
- enable distributed grid computing

Of these, the most common use is the creation of grids to join multiple centrally managed, partially autonomous, or fully autonomous clusters. The purpose of this section is to highlight the most common uses of grid technology and provide references to sections that further detail their configuration and management. Other sections cover the standard aspects of grid creation including configuring peer relationships, enabling data staging, credential management, usage policies, and other factors.

Image 23-1: Jobs Submitted to Grid Scheduler, Then Cluster Schedulers



23.1.3 Management-Scalability

Much like a massive-scalability cluster, a massive-scalability grid enables organizations to overcome scalability limitations in resource managers, networks, message passing libraries, security middleware, file systems, and other forms of software and hardware infrastructure. Moab does this by allowing a single large set of resources to be broken into multiple smaller, more manageable clusters, and then virtually re-assembling them using Moab. Moab becomes responsible for integrating the seams between the clusters and presenting a single-system image back to the end-users, admins, and managers.

Jobs cannot span clusters.

23.1.4 Resource Access

In some cases, the primary motivation for creating a grid is to aggregate resources of different types into a single system. This aggregation allows for multi-step jobs to run a portion of the job on one architecture, and a portion on another.

A common example of a multi-architecture parameter-sweep job is a batch regression test suite that requires a portion of the tests running on Redhat, a portion on SUSE, a portion on Myrinet nodes, and a portion on Infiniband nodes. While it would be very difficult to create and manage a single cluster that simultaneously provided all of these configurations, Moab can be used to create and manage a single grid that spans multiple clusters as needed.

23.1.5 Load-Balancing

While grids often have additional motivations, it is rare to have a grid created where increased total system utilization is not an objective. By aggregating the total pool of jobs requesting resources and increasing the pool of resources available to each job, Moab is able to improve overall system utilization, sometimes significantly. The biggest difficulty in managing multiple clusters is preventing inter-cluster policies and the cost of migration from overwhelming the benefits of decreased fragmentation losses. Even though remote resources may be available for immediate usage, migration costs can occur in the form of credential, job, or data staging and impose a noticeable loss in responsiveness on grid workload.

Moab provides tools to allow these costs to be monitored and managed and both cluster and grid level performance to be reported.

23.1.6 Single System Image (SSI)

Another common benefit of grids is the simplicity associated with a single system image-based resource pool. This simplicity generally increases productivity for end-users, admins, and managers.

An SSI environment tends to increase the efficiency of end-users by minimizing human errors associated with porting a request from a known system to a less known system. Additionally, the single point of access grid reduces human overhead associated with monitoring and managing workload within multiple independent systems.

For system admins, a single system image can reduce overhead, training time, and diagnostic time associated with managing a cluster. Furthermore, with Moab's peer-to-peer technology, no additional software layer is required to enable the grid and no new tools must be learned. No additional layers means no additional failure points, and that is good for everyone involved.

Managers benefit from SSI by being able to pursue organization mission objectives globally in a more coordinated and unified manner. They are also able to monitor progress toward those objectives and effectiveness of resources in general.

23.1.7 High Availability

A final benefit of grids is their ability to decrease the impact of failures. Grids add another layer of high availability to the cluster-level high availability. For some organizations, this benefit is a primary motivation, pulling together additional resources to allow workload to continue to be processed even in the event that some nodes, or even an entire cluster, become unavailable. Whether the resource unavailability is based on node failures, network failures, systems middleware, systems maintenance, or other factors, a properly configured grid can reroute priority workload throughout the grid to execute on other compatible resources.

With grids, there are a number of important factors in high availability that should be considered:

- enabling highly available job submission/job management interfaces
- avoiding network failures with redundant routes to compute resources
- handling partial failures
- dynamically restarting failed jobs

23.1.8 Grid Relationships

There are three types of relationships you can implement within the grid:

- Centralized Management (Moab Grid Control / Moab Grid Member)
- Centralized/Localized Management (Hierarchical)
- Localized Management (Peer-to-Peer)

23.1.8.A Centralized Management (Moab Grid Control / Moab Grid Member)

Note: Moab Grid Control / Moab Grid Member are also known as MGC/MGM.

The centralized management model (MGC/MGM) enables users to submit jobs to a centralized source server running Moab. The source Moab server obtains full resource information from all clusters and makes intelligent scheduling decisions across all clusters. Jobs (and data when configured to do so) are distributed to the remote clusters as needed. The centralized management model is recommended for intra-organization grid environments when cluster autonomy is not as necessary.

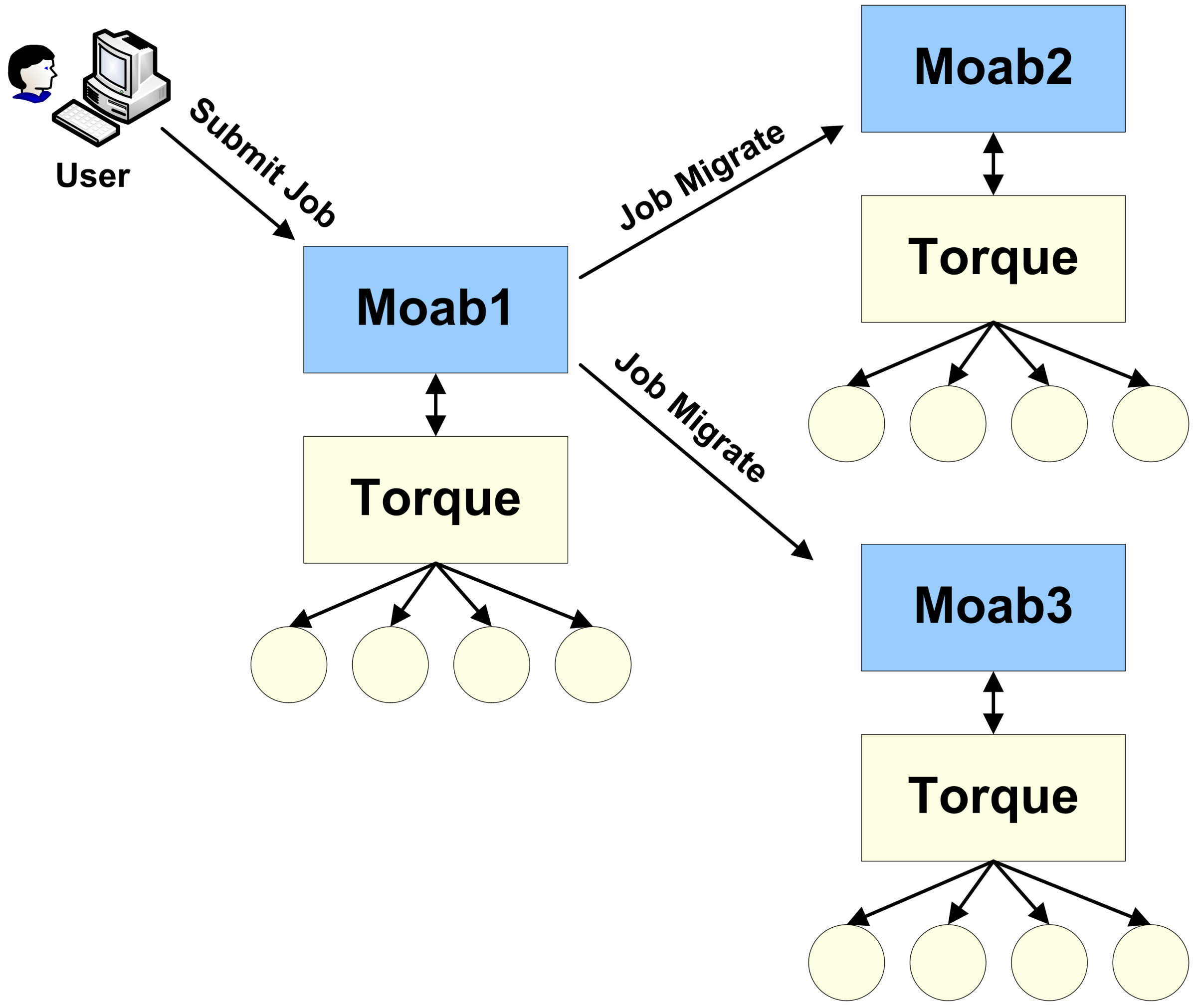
In the centralized management (Moab Grid Control / Moab Grid Member) configuration, roles are clear. In other configurations, individual Moab servers may simultaneously act as sources to some clusters and destinations to others or as both a source and a destination to another cluster.

Example of the Centralized Management (Moab Grid Control / Moab Grid Member) Model

XYZ Research has three clusters - MOAB1, MOAB2, and MOAB3--running Moab and the Torque resource manager. They want to submit jobs at a single location (cluster MOAB1) and have the jobs run on whichever cluster can provide the best responsiveness.

The desired behavior is essentially a *master-slave* relationship. MOAB1 is the central, or MGC cluster. On MOAB1, resource managers point to the local Torque resource manager and to the Moab servers on cluster MOAB2 and cluster MOAB3. The Moab servers on MOAB2 and MOAB3 are configured to trust cluster MOAB1 and to execute in SLAVE mode.

Image 23-2: Centralized Management



With this configuration, XYZ Research can submit jobs to the MGC Moab server running on cluster MOAB1 and can, as stated earlier, submit jobs from the MGM nodes also. However, only the MGC Moab server can schedule jobs. For example, cluster MOAB2 and cluster MOAB3 cannot schedule a job, but they can accept a job and retain it in an idle state until the MGC directs it to run.

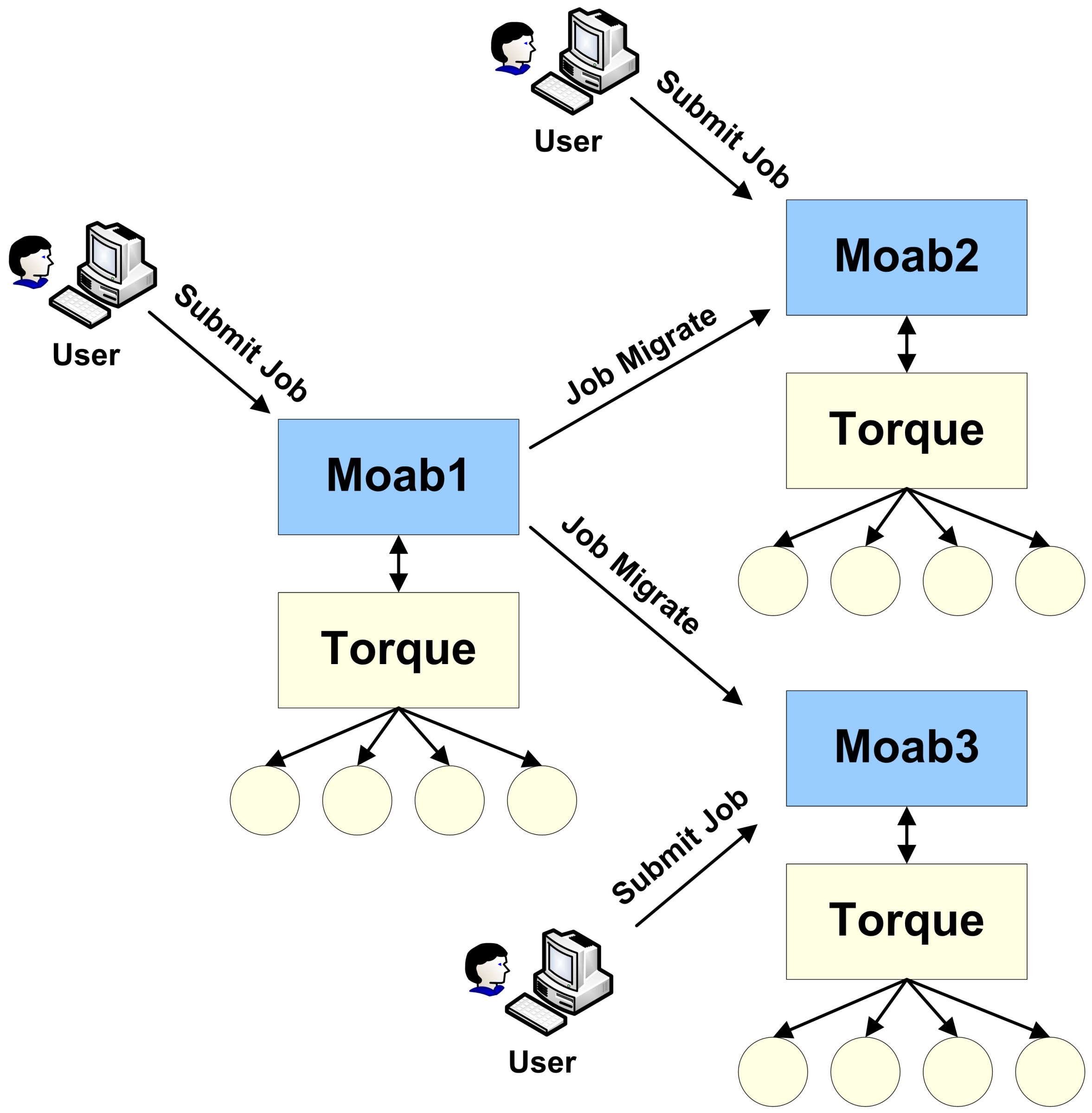
You can turn off job submission on MGM nodes by setting the DISABLESLAVEJOBSUBMIT parameter to `TRUE`.

the MGC Moab server obtains full resource information from all three clusters and makes intelligent scheduling decisions and distributes jobs (and data when configured to do so) to the remote clusters. The Moab servers running on clusters MOAB2 and MOAB3 are destinations behaving like a local resource manager. The Moab server running on MOAB1 is a source, loading and using this resource information.

23.1.8.B Centralized/Localized Management (Hierarchical)

As with the centralized management model (Moab Grid Control / Moab Grid Member), the hierarchical model enables users to submit jobs to a centralized source server running Moab. However, in the hierarchical model, clusters retain sovereignty, allowing local job scheduling. Therefore, if communication between the source and destination clusters is interrupted, the destination cluster(s) can still run jobs locally.

Image 23-3: Hierarchical Management

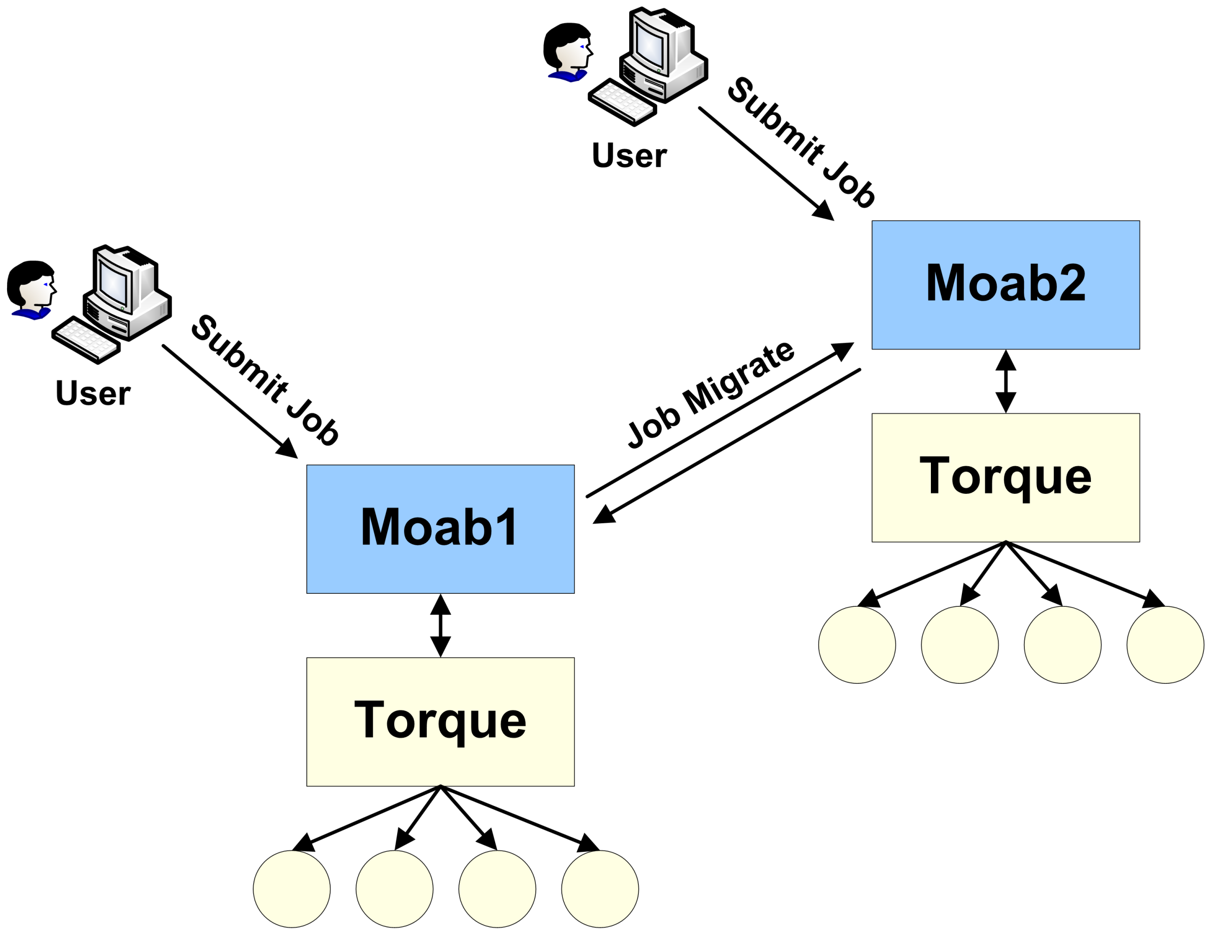


In the hierarchical model, the source Moab server obtains full resource information from all clusters and makes intelligent scheduling decisions across all clusters. As needed, jobs and data are distributed to the remote clusters. Or, if preferred, a destination cluster can also serve as its own source; however, a destination cluster cannot serve as a source to another destination cluster. The centralized management model is recommended for intra-organization grid environments when cluster autonomy and/or local management is necessary.

23.1.8.C Localized Management (Peer-to-Peer)

The localized management (peer-to-peer) model enables you to submit jobs on one cluster and schedule the jobs on the other cluster (it currently works with two clusters). For example, a job may be submitted on MOAB1 and run on MOAB2. Jobs can also migrate in the opposite direction (that is, from MOAB2 to MOAB1). The source servers running Moab obtain full resource information from both clusters and make intelligent scheduling decisions across both clusters. Jobs (and data when configured to do so) are migrated to other clusters as needed.

Image 23-4: Localized Management



Jobs will not migrate indefinitely. The localized management model limits them to one migration.

This model allows clusters to retain their autonomy while still allowing jobs to run on either cluster. No central location for job submission is needed, and you do not need to submit jobs from different nodes based on resource needs. You can submit a job from any location and it is either migrated to nodes on the least utilized cluster or the cluster requested in the job submission. This model is recommended for grids in an inter-organization grid environment.

23.1.9 Submitting Jobs to the Grid

In any peer-to-peer or grid environment where jobs must be migrated between clusters, use the Moab msub command. Once a job has been submitted to Moab using *msub*, Moab identifies potential destinations and migrates the job to the destination cluster.

Using Moab's *msub* job submission command, jobs can be submitted using PBS command file syntax and be run on any cluster using any of the resource managers. For example, a PBS job script can be submitted using *msub* and depending on availability, Moab can translate a subset of the job's directives and execute it on a PBS cluster.

Moab can only stage/migrate jobs between resource managers (in between clusters) that have been submitted using the *msub* command. If jobs are submitted directly to a low-level resource manager, such as PBS, Moab will still be able to schedule them, but only on resources directly managed by the resource manager to which they were submitted.

Example

A research lab wants to use spare cycles on its four clusters, each of which is running a local resource manager. In addition to providing better site-wide load balancing, the goal is to also provide some of its users with single point access to all compute resources. Various researchers have made it clear that this new multi-cluster load balancing must not impose any changes on users who are currently using these clusters by submitting jobs locally to each cluster.

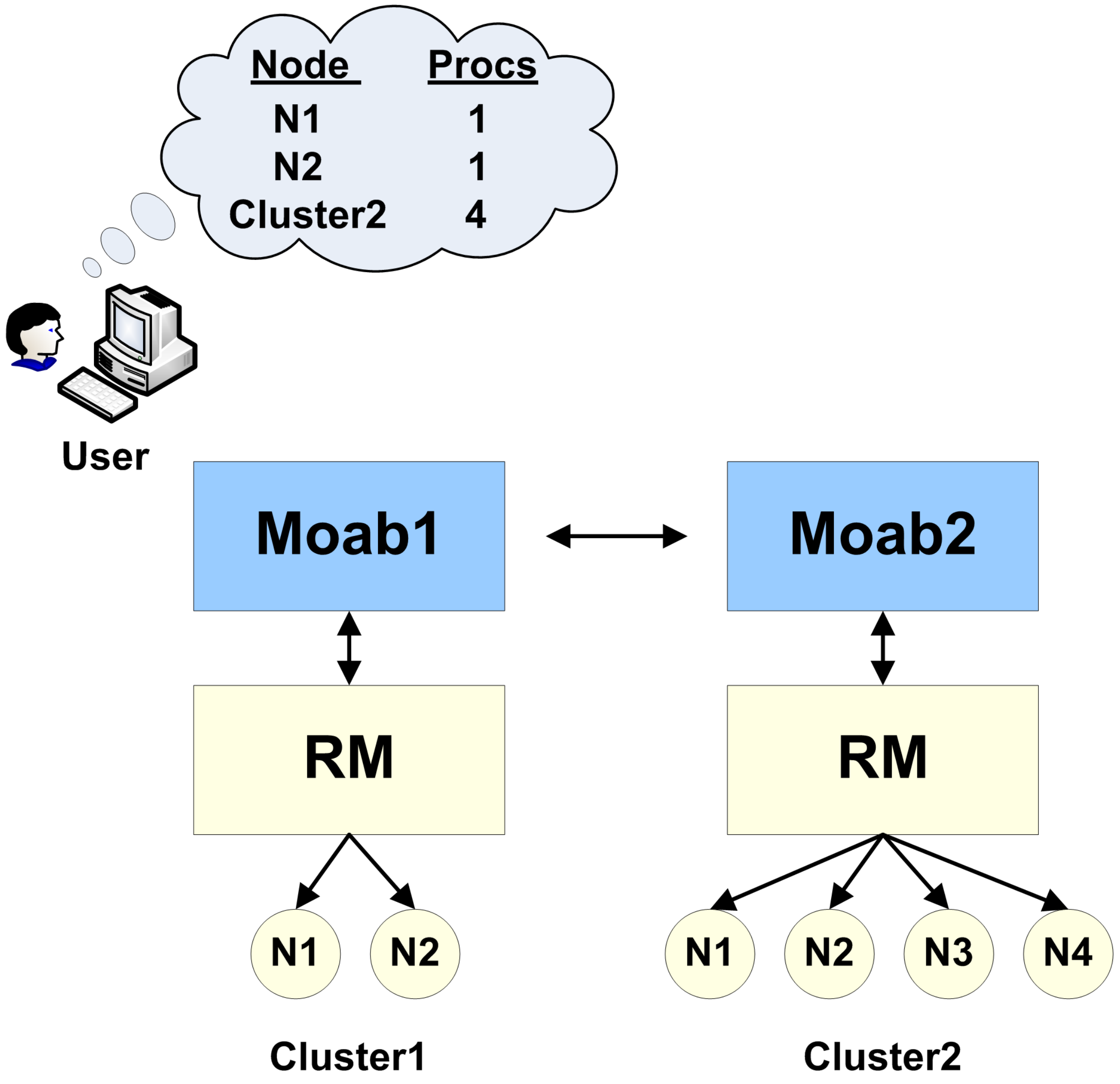
In this example, the scheduler mode of the destination clusters should be set to `NORMAL` rather than `SLAVE`. In `SLAVE` mode, Moab makes no local decisions - it simply follows the directions of remote trusted peers. In `NORMAL` mode, each Moab is fully autonomous, scheduling all local workload and coordinating with remote peers when and how to schedule migrated jobs.

From the perspective of a local cluster user, no new behaviors are seen. Remote jobs are migrated in from time to time, but to the user each job looks as if it were locally submitted. The user continues to submit, view, and manage jobs as before, using existing local jobs scripts.

23.1.10 Viewing Jobs and Resources

By default, each destination Moab server will report all compute nodes it finds back to the source Moab server. These reported nodes appear within the source Moab as local nodes each within a partition associated with the resource manager reporting them. If a source resource manager is named `slave1`, all nodes reported by it will be associated with the `slave1` partition. Users and admins communicating with the source Moab via Moab Cluster Manager, or standard Moab command line tools can view and analyze all reported nodes.

Image 23-5: Viewing Jobs and Resources



The grid view will be displayed if either the source or the destination server is configured with grid view.

For job information, the default behavior is to only report to the source Moab information regarding jobs that originated at the source. If information about other jobs is desired, this can be configured as shown in the section Workload Submission and Control.

Related Topics

- 23.6 Resource Control and Access

23.2 Grid Configuration Basics

In this section:

23.2.1 Peer Configuration

In the simplest case, establishing a peer relationship can be accomplished with as few as two configuration lines: one line to indicate how to contact the peer and one line to indicate how to authenticate the server. However, data migration issues, credential mapping, and usage policies must often be addressed in order to make a peer-based grid effective.

To address these issues Moab provides facilities to control how peers inter-operate, enabling full autonomy over both client and server ends of the peer relationship.

23.2.2 Initial Configuration

At a minimum, only two parameters must be specified to establish a peer relationship: RMCFG and CLIENTCFG[<X>]. `RMCFG` enables you to specify interface information directing Moab on how to contact and inter-operate with the peer. For peer interfaces, a few guidelines must be followed with the `RMCFG` parameter:

- The `TYPE` attribute of the peer must be set to `moab`.
- The `SERVER` attribute must point to the host and user interface port of the remote Moab server.

- The `name` of the resource manager should match the name of the remote peer cluster as specified with the SCHEDCFG parameter in the peer `moab.cfg`.

```
# moab.cfg on MoabServer01

SCHEDCFG[MoabServer01] MODE=NORMAL SERVER=hpc-01:41111
RMCFG[MoabServer02]    TYPE=moab   SERVER=hpc-02:40559
...
```

Configuring the `CLIENTCFG` parameter is mandatory. When specifying the `CLIENTCFG` parameter for peers, the following guidelines must be followed:

- The `CLIENTCFG` parameter must be specified in the `moab-private.cfg` file on both peers.
- A `RM:` prefix is required before the peer's name.
- If using default secret key based security, the value of the `KEY` attribute must match the `KEY` value set on the corresponding remote peer.
- The `AUTH` attribute must be set to `admin1` in the `moab-private.cfg` on the destination Moab.

```
# moab-private.cfg on MoabServer01

CLIENTCFG[RM:MoabServer02] KEY=3esfv0=32re2-tdbne
....
```

```
# moab-private.cfg on MoabServer02

CLIENTCFG[RM:MoabServer01] KEY=3esfv0=32re2-tdbne AUTH=admin1
...
```

23.3 Centralized Grid Management (Moab Grid Control / Moab Grid Member)

In this section:

23.3.1 Moab Grid Control Configuration

Note: Moab Grid Control / Moab Grid Member are also known as MGC/MGM.

The process of setting up the Moab Grid Control Configuration is the same as setting up a source Moab configuration. The Moab Grid Control / Moab Grid Member relationship is configured in each `moab.cfg` on the MGM.

```
# moab.cfg on Master

SCHEDCFG[master] SERVER=master:42559 MODE=NORMAL
...
```

```
# moab-private.cfg on Master

CLIENTCFG[RM:slave1]  KEY=3esfv0=32re2-tdbne
...
```

23.3.2 Moab Grid Member Configuration

the MGM's relationship with the MGC is determined by the `MODE`. Setting `MODE` to `SLAVE` notifies the MGC to take control of starting jobs on the MGM. the MGC starts the jobs on the MGM. In `SLAVE` mode, jobs can be submitted locally to the MGM, but are not seen or started by the MGC. When a job is submitted locally to the MGM, the job is locked into the cluster and cannot migrate to other clusters.

```
# moab.cfg on Slave

SCHEDCFG[slave1] SERVER=slave1:42559 MODE=SLAVE
...
```

```
# moab-private.cfg on Slave

CLIENTCFG[RM:master] KEY=3esfv0=32re2-tdbne  AUTH=admin1
...
```

23.4 Hierarchical Grid Management

In this section:

23.4.1 Configuring a Peer Server (Source)

Peer relationships are enabled by creating and configuring a resource manager interface using the RMCFG parameter. This interface defines how a given Moab will load resource and workload information and enforce its scheduling decisions. In non-peer cases, the `RMCFG` parameter points to a resource manager such as Torque. However, if the `TYPE` attribute is set to `moab`, the `RMCFG` parameter can be used to configure and manage a peer relationship.

23.4.2 Simple Hierarchical Grid

The first step to create a new peer relationship is to configure an interface to a destination Moab server. In the following example, cluster `C1` is configured to be able to *see* and *use* resources from two other clusters:

```
SCHEDCFG[C1] MODE=NORMAL SERVER=head.C1.xyz.com:41111
RMCFG[C2]    TYPE=moab   SERVER=head.C2.xyz.com:40559
RMCFG[C3]    TYPE=moab   SERVER=head.C3.xyz.com:40559
...
```

`C1` allows a global view of the underlying clusters. From `C1`, jobs can be viewed and modified. `C2` and `C3` act as separate scheduling entities that can receive jobs from `C1`. `C1` migrates jobs to `C2` and `C3` based on available resources and policies of `C1`. Jobs migrated to `C2` and `C3` are scheduled according to the polices on `C2` and `C3`.

In this case, one `RMCFG` parameter is all that is required to configure each peer relationship if standard secret key based authentication is being used and a shared default secret key exists between the source and destination Moabs. However, if peer relationships with multiple clusters are to be established and a per-peer secret key is to be used (highly recommended), then a CLIENTCFG parameter must be specified for the authentication mechanism. Because the secret key must be kept secure, it must be specified in the `moab-private.cfg` file. For the current example, a per-peer secret key could be set up by creating the following `moab-private.cfg` file on the `C1` cluster:

```
CLIENTCFG[RM:C2] KEY=fastclu3t3r
CLIENTCFG[RM:C3] KEY=14436aaa
```

The key specified can be any alphanumeric value and can be locally generated or made up. The only critical aspect is that the keys specified on each end of the peer relationship match.

Additional information can be found in the Grid Security section, which provides detailed information on designing, configuring, and troubleshooting peer security.

Continuing with the example, the initial source side configuration is now complete. On the destination clusters, `C2` and `C3`, the first step is to configure authentication. If a shared default secret key exists between all three clusters, then configuration is complete and the clusters are ready to communicate. If per-peer secret keys are used (recommended), then it will be necessary to create matching `moab-private.cfg` files on each of the destination clusters. With this example, the following files are required on `C2` and `C3` respectively:

```
CLIENTCFG[RM:C1] KEY=fastclu3t3r AUTH=admin1
```

```
CLIENTCFG[RM:C1] KEY=14436aaa AUTH=admin1
```

Once peer security is established, a final optional step is to configure scheduling behavior on the destination clusters. By default, each destination cluster accepts jobs from each trusted peer. However, it will also be fully autonomous, accepting and scheduling locally submitted jobs and enforcing its own local policies and optimizations. If this is the desired behavior, then configuration is complete.

In the current example, with no destination side scheduling configuration, jobs submitted to cluster `C1` can run locally on cluster `C2` or on cluster `C3`. However, the established configuration does not necessarily enforce a strict master-slave relationship because each destination cluster (`C2` and `C3`) has complete autonomy over how, when, and where it schedules both local and remote jobs. Each cluster can potentially receive jobs that are locally submitted and can also receive jobs from other source Moab servers.

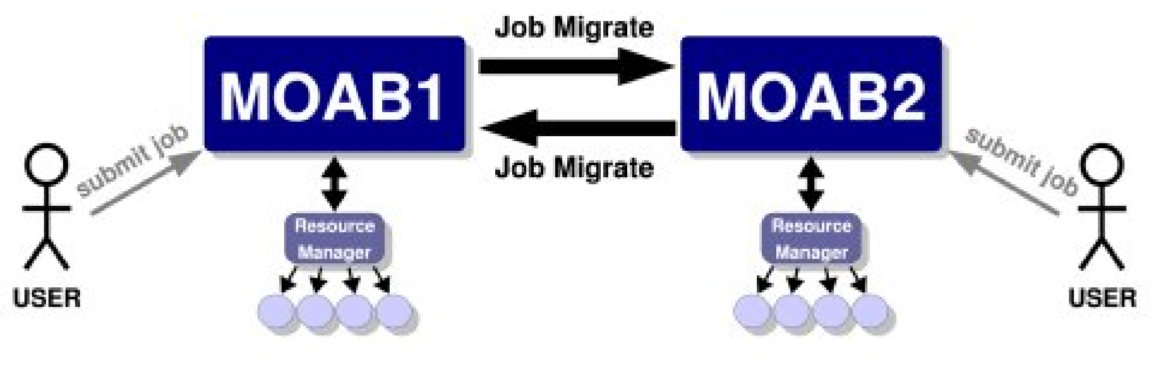
Further, each destination cluster will accept any and all jobs migrated to it from a trusted peer without limitations on who can run, when and where they can run, or how many resources they can use. If this behavior is either too restrictive or not restrictive enough, then destination side configuration will be required.

23.5 Localized Grid Management

In this section:

23.5.1 Enabling Bi-Directional Job Flow

Image 23-6: Bi-directional peer-to-peer setup



For each peer interface, a RMCFG parameter is only required for the server (or source side of the interface). If two peers are to share jobs in both directions, the relationship is considered to be bi-directional.

23.5.2 True Peer-to-Peer Grid

Previous examples involved grid masters that coordinated the activities of the grid and made it so direct contact between peers was not required. However, if preferred, the master is not required and individual clusters can interface directly with each other in a true peer manner. This configuration is illustrated in the following example:

```
# Cluster A

SCHEDCFG[clusterA] MODE=NORMAL SERVER=clusterA
RMCFG[clusterA]    TYPE=pbs
RMCFG[clusterB]    TYPE=moab   SERVER=clusterB:40559
CLIENTCFG[RM:clusterB] AUTH=admin1 KEY=banana16
```

```
# Cluster B

SCHEDCFG[clusterB] MODE=NORMAL SERVER=clusterB
RMCFG[clusterB]    TYPE=pbs
RMCFG[clusterA]    TYPE=moab   SERVER=clusterA:40559
CLIENTCFG[RM:clusterA] AUTH=admin1 KEY=banana16
```

If you are using Moab Accounting Manager, the `Start` action is not supported as a non-blocking accounting action in Peer-to-Peer grids. You will need to include `Start` as a blocking action. For example:

```
AMCFG[mam] BLOCKINGACTIONS=Start
```

23.6 Resource Control and Access

In this section:

23.6.1 Controlling Resource Information

In a Moab peer-to-peer grid, resources can be viewed in one of two models:

- Direct - nodes are reported to remote clusters exactly as they appear in the local cluster
- Mapped - nodes are reported as individual nodes, but node names are mapped to a unique name when imported into the remote cluster

23.6.1.A Direct Node View

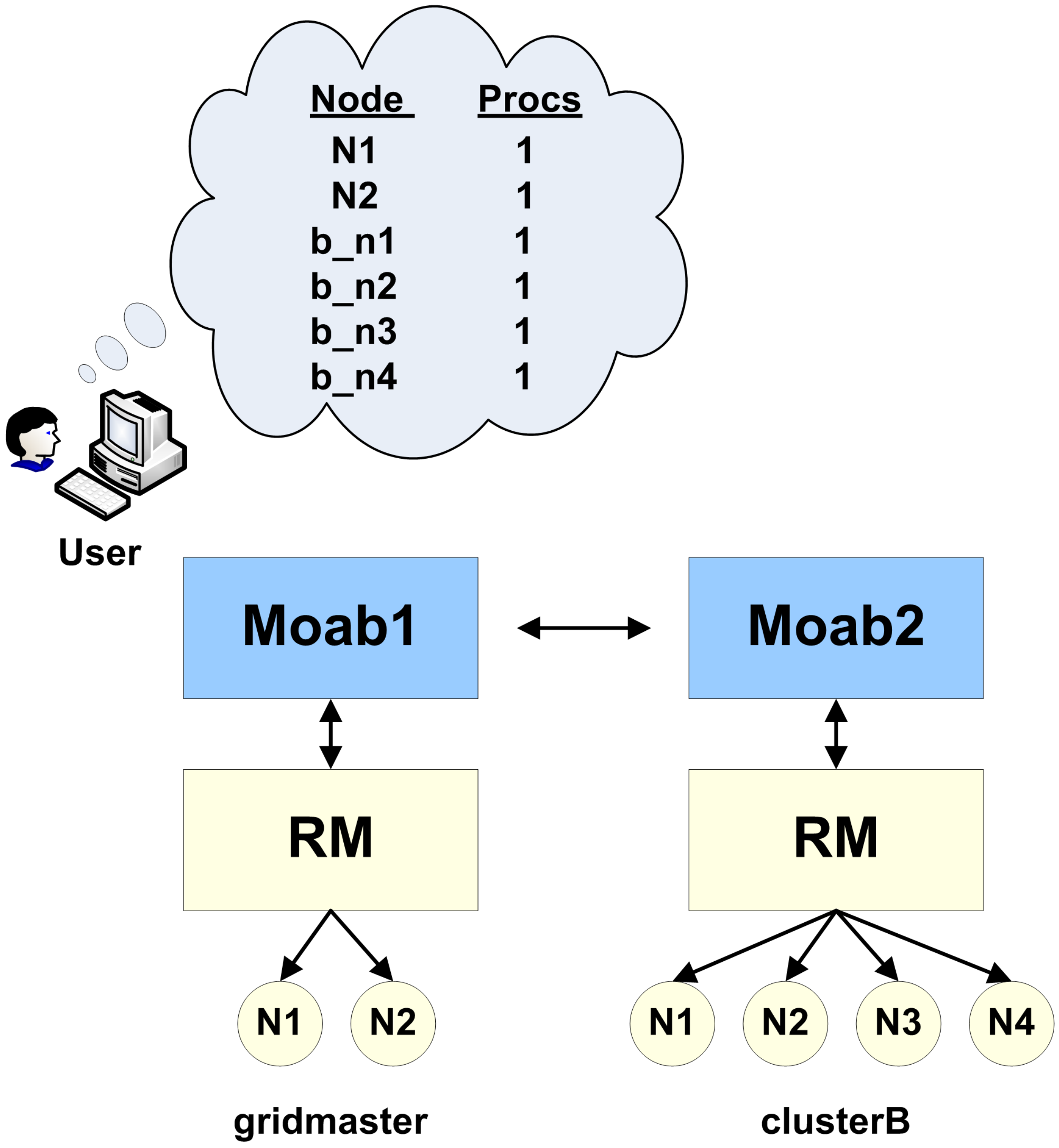
Direct node import is the default resource information mode. No additional configuration is required to enable this mode.

23.6.1.B Mapped Node View

In this mode, nodes are reported just as they appear locally by the exporting cluster. However, on the importing cluster side, Moab maps the specified node names using the resource manager object map. In an object map, node mapping is specified using the `node` keyword as in the following example:

```
SCHEDCFG[gridmaster] MODE=NORMAL
RMCFG[clusterB]      TYPE=moab OMAP=file://$HOME/clusterb.omap.dat
...
node:b_*,*
```

Image 23-7: Mapped Node View

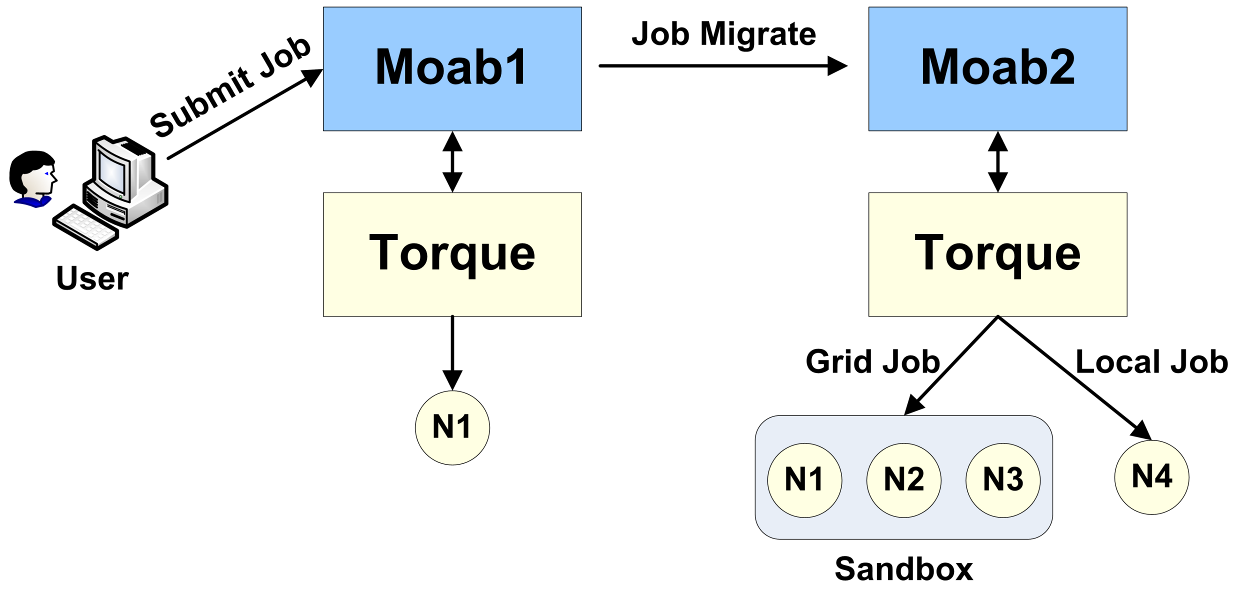


In this example, all nodes reported by `clusterB` have the string `b_` prepended to prevent node name space conflicts with nodes from other clusters. For example, if cluster `clusterB` reported the nodes `node01`, `node02`, and `node03`, cluster `gridmaster` reports them as `b_node01`, `b_node02`, and `b_node03`.

See 23.11.2 Peer Credential Mapping for more information on creating an object map file.

23.6.2 Managing Resources with Grid Sandboxes

Image 23-8: Grid Sandbox



A cluster may want to participate in a grid but may desire to dedicate only a set amount of resources to external grid workload or may only want certain peers to have access to particular sets of resources. With Moab, this can be achieved by way of a grid sandbox, which must be configured at the destination cluster. Grid sandboxes can both constrain external resource access and limit which resources are reported to other peers. This allows a cluster to only report a defined subset of its total resources to source peers and restricts peer workload to the sandbox. The sandbox can be set aside for peer use exclusively, or can allow local workload to also run inside of it. Through the use of multiple, possibly overlapping grid sandboxes, you can fully control resource availability on a per peer basis.

A grid sandbox is created by configuring a standing reservation on a destination peer and then specifying the `ALLOWGRID` flag on that reservation. This flag tells the Moab destination peer to treat the standing reservation as a grid sandbox, and, by default, only the resources in the sandbox are visible to grid peers. Also, the sandbox only allows workload from other peers to run on the contained resources.

Example 23-1: Dedicated Grid Sandbox

```
SRCFG[sandbox1] PERIOD=INFINITY HOSTLIST=node01,node02,node03
SRCFG[sandbox1] CLUSTERLIST=ALL FLAGS=ALLOWGRID
...
```

The standing reservation `sandbox1` creates a grid sandbox, which always exists and contains the nodes `node01`, `node02`, and `node03`. This sandbox will only allow grid workload to run within it by default. This means that the scheduler will not consider the boxed resources for local workload.

Grid sandboxes inherit all of the same power and flexibility that standing reservations have. See 6.1.5 Configuring and Managing Reservations for additional information.

> The flag `ALLOWGRID` marks the reservation as a grid sandbox and as such, it precludes grid jobs from running anywhere else. However, it does *not* enable access to the reserved resources. The `CLUSTERLIST` attribute in the above example enables access to all remote jobs.

Controlling Access on a Per Cluster Basis

Often clusters may want to control which peers are allowed to use certain sandboxes. For example, Cluster A may have a special contract with Cluster B and will let overflow workload from Cluster B run on 60% of its resources. A third peer in the grid, Cluster C, doesn't have the same contractual agreement, and is only allowed 10% of Cluster A at any given time. Therefore two separate sandboxes must be made to accommodate the different policies.

```
SRCFG[sandbox1] PERIOD=INFINITY HOSTLIST=node01,node02,node03,node04,node05
SRCFG[sandbox1] FLAGS=ALLOWGRID CLUSTERLIST=ClusterB
SRCFG[sandbox2] PERIOD=INFINITY HOSTLIST=node06 FLAGS=ALLOWGRID
SRCFG[sandbox2] CLUSTERLIST=ClusterB,ClusterC,ClusterD USERLIST=ALL
...
```

This example configuration illustrates how cluster A could set up their sandboxes to follow a more complicated policy. In this policy, `sandbox1`provides exclusive access to nodes 1 through 5 to jobs coming from peer ClusterB by including `CLUSTERLIST=ClusterB` in the definition. Reservation `sandbox2`provides shared access to `node6` to local jobs and to jobs from clusters B, C, and D through use of the `CLUSTERLIST` and `USERLIST` attributes.

With this setup, the following policies are enforced:

- local jobs can see all nodes and run anywhere except nodes 1 through 5
- jobs from cluster B can see and run only on nodes 1 through 6
- jobs from clusters C and D can see and run only on node 6

As shown in the example above, sandboxes can be shared across multiple peers by listing all sharing peers in the `CLUSTERLIST` attribute (comma-delimited).

Access Control Lists/Granting Access to Local Jobs

It is not always desirable to have the grid sandbox reserve resources for grid consumption, exclusively. Many clusters may want to use the grid sandbox when local workload is high and demand from the grid is relatively low. Clusters may also want to further restrict what kind of grid workload can run in a sandbox. This fine-grained control can be achieved by attaching access control lists (ACLs) to grid sandboxes.

Since sandboxes are basically special standing reservations, the syntax and rules for specifying an ACL is identical to those found in 6.1.5 Configuring and Managing Reservations.

Example

```
SRCFG[sandbox2] PERIOD=INFINITY HOSTLIST=node04,node05,node06
SRCFG[sandbox2] FLAGS=ALLOWGRID QOSLIST=high GROUPLIST=engineer
...
```

A cluster decides to dedicate resources to a sandbox, but wants local workload to also run within it. An additional ACL is then associated with the definition. The reservation `sandbox2` takes advantage of this feature by allowing local jobs running with a QOS of `high`, or under the group `engineer`, to also run on the sandboxed nodes `node04`, `node05`, and `node06`.

23.7 Workload Submission and Control

Controlling Peer Workload Information

By default, a peer is only responsible for workload that is submitted via that particular peer. This means that when a source peer communicates with destination peers it only receives information about workload it sent to those destination peers. If desired, the destination peers can send information about *all* of its workload: both jobs originating locally and remotely. This is called *local workload exporting*. This might help simplify administration of different clusters by centralizing monitoring and management of jobs at one peer.

To implement local workload exporting, use the `LOCALWORKLOADEXPORT` resource manager flag. For example:

```
RMCFG[ClusterA.INBOUND] FLAGS=LOCALWORKLOADEXPORT   # source peer
...
```

This example shows the configuration on a destination peer (ClusterB) that exports its local and remote workload to the source peer (ClusterA).

LOCALWORKLOADEXPORT does not need to be configured in Moab Grid Control / Moab Grid Member grids.

Related Topics

- 13.4 Job Start Time Estimates

23.8 Reservations in the Grid

In some environments, globally-shared resources might need to be managed to guarantee the full environment required by a particular job. Resources such as networks, storage systems, and license managers can be used only by batch workload but this workload can be distributed among multiple independent clusters. Consequently, the jobs from one cluster can utilize resources required by jobs from another. Without a method of coordinating the needs of the various cluster schedulers, resource reservations will not be respected by other clusters and will be of only limited value.

Using the centralized model, Moab allows the importing and exporting of reservations from one peer server to another. With this capability, a source peer can be set up for the shared resource to act as a clearinghouse for other Moab cluster schedulers. This source peer Moab server reports configured and available resource state and in essence possesses a global view of resource reservations for all clusters for the associated resource.

To allow the destination peer to export reservation information to the source Moab, the RMCFG lines for all client resource managers must include the flag `RSVEXPORT`. The source Moab should be configured with a resource manager interface to the destination peer and include both the `RSVEXPORT` and `RSVIMPORT` flags. For the destination peer, `RSVEXPORT` indicates that it should *push* information about newly created reservations to the source Moab, while the `RSVIMPORT` flag indicates that the source Moab server should import and locally enforce reservations detected on the destination peer server.

23.9 Grid Usage Policies

In this section:

23.9.1 Grid Usage Policy Overview

Moab allows extensive control over how peers interact. These controls allow the following:

- Limiting which remote users, group, and accounts can utilize local compute resources
- Limiting the total quantity of local resources made available to remote jobs at any given time
- Limiting remote resource access to a specific subset of resources
- Limiting timeframes during which local resources will be made available to remote jobs
- Limiting the types of remote jobs that will be allowed to execute

23.9.2 Peer Job Resource Limits

Both source and destination peers can limit the types of jobs they will allow in terms of resources requested, services provided, job duration, applications used, etc using Moab's job template feature. Using this method, one or more job profiles can be created on either the source or destination side, and Moab can be configured to allow or reject jobs based on whether or not the jobs meet the specified job profiles.

When using the `ALLOWJOBLIST` and `REJECTJOBLIST` attributes, the following rules apply:

- All jobs that meet the job templates listed by `ALLOWJOBLIST` are allowed.
- All jobs that do not meet `ALLOWJOBLIST` job templates and which do meet `REJECTJOBLIST` job templates are rejected.
- All jobs that meet no job templates in either list are allowed.

23.9.3 Usage Limits via Peer Credentials

With peer interfaces, destination clusters willing to accept remote jobs can map these jobs onto a select subset of users, accounts, QoSes, and queues. With the ability to lock these jobs into certain credentials comes the ability to apply any arbitrary credential constraints, priority adjustments, and resource limitations normally available within cluster management. Specifically, the following can be accomplished:

- Limit number of active jobs simultaneously allowed
- Limit quantity of allocated compute resources simultaneously allowed
- Adjust job priority

- Control access to specific scheduling features (deadlines, reservations, preemption, etc.)
- Adjust fairshare targets
- Limit resource access

23.9.4 Using General Policies in a Grid Environment

While Moab does provide a number of unique grid-based policies for use in a grid environment, the vast majority of available management tools come from the transparent application of cluster policies. Cluster-level policies such as job prioritization, node allocation, fairshare, usage limits, reservations, preemption, and allocation management all just work and can be applied in a grid in exactly the same manner.

The one key concept to understand is that in a centralized based grid, these policies apply across the entire grid; in a peer-based grid, these policies apply only to local workload and resources.

Source Cluster Policies

In many cases, organizations are interested in treating jobs differently based on their point of origin. This can be accomplished by assigning and/or keying off of a unique credential associated with the remote workload. For example, you might want to constrain jobs from a remote cluster to only a portion of the total available cluster cycles. This could be accomplished using usage limits, fairshare targets, fairshare caps, reservations, or allocation management based policies.

The examples below show three different approaches for constraining remote resource access.

Example 23-2: Constraining Remote Resource Access via Fairshare Caps

```
# define peer relationship and map all incoming jobs to orion account
RMCFG[orion.INBOUND] SET.JOB=orion.set
JOBCFG[orion.set] ACCOUNT=orion
# configure basic fairshare for 7 one day intervals
FSPOLICY DEDICATEDPS
FSINTERVAL 24:00:00
FSDEPTH 7
FSUSERWEIGHT 100
# use fairshare cap to limit jobs from orion to 10% of cycles
ACCOUNTCFG[orion] FSCAP=10%
```

Example 23-3: Constraining Remote Resource Access via Fairshare Targets and Preemption

```
# define peer relationship and map all incoming jobs to orion account
RMCFG[orion.INBOUND] SET.JOB=orion.set
JOBCFG[orion.set] ACCOUNT=orion
# local cluster can preempt jobs from orion
USERCFG[DEFAULT] JOBFLAGS=PREEMPTOR
```

```
PREEMPTPOLICY CANCEL
# configure basic fairshare for 7 one day intervals
FSPOLICY DEDICATEDPS
FSINTERVAL 24:00:00
FSDEPTH 7
FSUSERWEIGHT 100
# decrease priority of remote jobs and force jobs exceeding 10% usage to be
preemptible
ACCOUNTCFG[orion] FSTARGET=10-
ENABLEFSVIOLATIONPREEMPTION TRUE
```

Example 23-4: Constraining Remote Resource Access via Priority and Usage Limits

```
# define peer relationship and map all incoming jobs to orion account RMCFG
[orion.INBOUND] SET.JOB=orion.set
JOBCFG[orion.set] QOS=orion
USERCFG[DEFAULT] QDEF=orion
# local cluster can preempt jobs from orion
USERCFG[DEFAULT] JOBFLAGS=PREEMPTOR
PREEMPTPOLICY CANCEL
# adjust remote jobs to have reduced priority
QOSCFG[orion] PRIORITY=-1000
# allow remote jobs to use up to 64 procs without being preemptible and up to 96 as
preemptees
QOSCFG[orion] MAXPROC=64,96
ENABLESPVIOLATIONPREEMPTION TRUE
```

Related Topics

- 23.6.2 Managing Resources with Grid Sandboxes - control grid resource access

23.10 Grid Scheduling Policies

In this section:

23.10.1 Peer-to-Peer Resource Affinity

The concept of resource affinity stems from a number of facts:

- Certain compute architectures are able to execute certain compute jobs more effectively than others.

- From a given location, staging jobs to various clusters might require more expensive allocations, more data and network resources, and more use of system services.
- Certain compute resources are owned by external organizations and should be used sparingly.

Regardless of the reason, Moab servers allow the use of peer resource affinity to guide jobs to the clusters that make the best fit according to a number of criteria.

At a high level, this is accomplished by creating a number of job templates and associating the profiles with different peers with varying impacts on estimated execution time and peer affinity.

23.10.2 Peer Allocation Policies

A direct way to assign a peer allocation algorithm is with the PARALLOCATIONPOLICY parameter. Values are listed in the following table:

Value	Description
FirstStart	Allocates resources from the eligible peer that can start the job the soonest.
LoadBalance	Allocates resources from the eligible peer with the most available resources; measured in tasks (balances workload distribution across potential peers).
LoadBalanceP	Allocates resources from the eligible peer with the most available resources; measured in percent of configured resources (balances workload distribution across potential peers).
Random	Allocates partitions in a random order each iteration. In general, all the jobs scheduled within the same iteration receive the same randomized list of partitions. This means the randomization happens between iterations and not within the same iteration. One iteration Moab might start with partition X and the next it might start with partition Y.
RoundRobin	Allocates resources from the eligible peer that has been least recently allocated.

The mdiag -t -v command can be used to view current calculated partition priority values.

23.10.3 Per-partition Scheduling

Per-partition scheduling can be enabled by adding the following lines to `moab.cfg`:

```
PERPARTITIONSCHEDULING TRUE
JOBMIGRATEPOLICY JUSTINTIME
```

To use per-partition scheduling, you must configure fairshare trees where particular users have higher priorities on one partition, and other users have higher priorities on a different partition.

Do not set the USEANYPARTITIONPRIO parameter if you use per-partition scheduling. Doing so causes Moab to schedule jobs to the first partition listed, even if nodes from another partition will be available sooner.

23.11 Grid Credential Management

In this section:

23.11.1 Peer Credential Management

Moab provides a number of credential management features that allow sites to control which local users can utilize remote resources and which remote users can utilize local resources and under what conditions this access is granted.

23.11.2 Peer Credential Mapping

If two peers share a common user space (a given user has the same login on both clusters), then there is often no need to enable credential mapping. When users, groups, classes, QoSes, and accounts are not the same from one peer to another, Moab enables you to specify an Object Map URL. This URL contains simple one to one or expression based mapping for credentials and other objects. Using the RMCFG parameter's OMAP attribute, you can tell Moab where to find these mappings. The object map uses the following format:

```
<OBJECTTYPE>:<SOURCE_OBJECTID>,<DESTINATION_OBJECTID>
```

Where <SOURCE_OBJECT> can be a particular username or an asterisk (*) that is a wildcard matching all credentials of the specified type, which have not already been matched.

The object map file can be used to translate the following:

Keyword	Objects
account	accounts/projects
class	classes/queues
file	files/directories
group	groups
node	nodes
qos	QoS
user	users

The following `moab.cfg` and `omap.dat` files demonstrate a sample credential mapping:

```
SCHEDCFG[master1] MODE=normal
RMCFG[slave1]     OMAP=file:///opt/moab/omap.dat
...
```

```
user:joe,jsmith
user:steve,sjohnson
group:test,staff
class:batch,serial
user:*,grid
```

In this example, a job that is being migrated from cluster `master1` to the peer `slave1` will have its credentials mapped according to the contents of the `omap.dat` file. In this case, a job submitted by user `joe` on `master1` will be executed under the user account `jsmith` on peer `slave1`. Any credential that is not found in the mapping file will be passed to the peer as submitted. In the case of the user credential, all users other than `joe` and `steve` will be remapped to the user `grid` due to the wildcard matching.

Because the `OMAP` attribute is specified as a URL, multiple methods can be used to obtain the mapping information. In addition to the file protocol shown in the example above, exec can be used.

Note that there is no need to use the credential mapping facility to map all credentials. In some cases, a common user space exists but it is used to map all classes/queues on the source side to a single queue on the destination side. Likewise, for utilization tracking purposes, it may be desirable to map all source account credentials to a single cluster-wide account.

23.11.3 Source and Destination Side Credential Mapping

Credential mapping can be implemented on the source cluster, destination cluster, or both. A source cluster may want to map all user names for all outgoing jobs to the name `generaluser` for security purposes, and a destination cluster may want to remap all incoming jobs from this particular user to the username `cluster2` and the QoS `grid`.

23.11.4 Preventing User Space Collisions

In some cases, a cluster might receive jobs from two independent clusters where grid wide username distinctiveness is not guaranteed. In this case, credential mapping can be used to ensure the uniqueness of each name. With credential mapping files, this can be accomplished using the `<DESTINATION_CREDENTIAL>` wildcard asterisk (*) character. If specified, this character will be replaced with the exact `<SOURCE_CREDENTIAL>` when generating the destination credential string. For example, consider the following configuration:

```
SCHEDCFG[master1] MODE=normal
RMCFG[slave1]     OMAP=file:///opt/moab/omap.dat  FLAGS=client
...
```

```
user:*,c1_*
group:*,*_grid
account:*,temp_*
```

This configuration will remap the usernames of all jobs coming in from the peer `slave1`. The username `john` will be remapped to `c1_john`, the group `staff` will be remapped to `staff_grid` and the account `demo` will be remapped to `temp_demo`.

23.12 Grid Data Management

This method of data staging has been deprecated and will be removed in a future release. See 24.1 Data Staging Example for information about the new method of staging data.

In this section:

23.12.1 Grid Data Management Overview

Moab provides a highly generalized data manager interface that can allow both simple and advanced data management services to be used to migrate data amongst peer clusters. Using a flexible script interface, services such as *scp*, *NFS*, and *gridftp* can be used to address data staging needs. This feature enables a Moab peer to push job data to a destination Moab peer.

23.12.2 Configuring Peer Data Staging

Moab offers a simple, automatic configuration, and advanced configuration options. At a high level, configuring data staging across a peer-to-peer relationship consists of configuring one or more storage managers, associating them with the appropriate peer resource managers, and then specifying data requirements at the local level—when the job is submitted.

To use the data staging features, you must specify the `--with-grid` option at `./configure` time. After properly configuring data staging, you can submit a job to the peer with any user who has SSH keys set up and Moab will automatically or implicitly stage back the standard out and standard error files created by the job. Files can be implicitly staged in or out before a job runs by using the mstagein or mstageout options of msub.

Simple Configuration

Moab automatically does most of the data staging configuration based on a simplified set of parameters (most common defaults) in the configuration file (`moab.cfg`).

Do the following to configure peer data staging:

1. Configure at least two Moab clusters to work in a grid. Refer to information throughout Chapter 23: Moab Workload Manager for Grids for help on configuring Moab clusters to work together as peers in a grid.
2. Set up SSH keys so that users on the source grid peer can SSH to destination peers without the need for a password.

3. Make necessary changes to the `moab.cfg` file of the source grid peer to activate data staging, which involves creating a new data resource manager definition within Moab. The resource manager provides data staging services to existing peers in the grid. By defining the data resource manager within the moab.cfg, Moab automatically sets up all of the necessary data staging auxiliary scripts.

   Use the following syntax for defining a data resource manager:

   ```
   RMCFG[<RMName>] TYPE=NATIVE RESOURCETYPE=STORAGE
   VARIABLES=DATASPACEUSER=<DataSpaceUser>,DATASPACEDIR=<DataSpaceDir>
   SERVER=<DataServer>
   ```

   - `<RMName>`: Name of the resource manager (defined as a storage RM type by RESOURCETYPE=STORAGE).
   - `<DataSpaceUser>`: User used to SSH into `<DataServer>` to determine available space in `<DataSpaceDir>`. Moab runs a command similar to the following: `ssh <DataServer> -l <DataSpaceUser> df <DataSpaceDir>`
   - `<DataSpaceDir>`: Directory where staged data is stored.
   - `<DataServer>`: Name of the server, where `<DataSpaceDir>` is located.

   Define the following URLs:

   ```
   RMCFG[data] CLUSTERQUERYURL=exec://$TOOLSDIR/grid/cluster.query.dstage.pl
   RMCFG[data] SYSTEMMODIFYURL=exec://$TOOLSDIR/grid/system.modify.dstage.pl
   RMCFG[data] SYSTEMQUERYURL=exec://$TOOLSDIR/grid/system.query.dstage.pl
   RMCFG[data] RMINITIALIZEURL=exec://$TOOLSDIR/grid/setup.config.pl
   ```

4. Associate the data resource manager with a peer resource manager:

   ```
   RMCFG[remote_data] TYPE=NATIVE RESOURCETYPE=STORAGE
   VARIABLES=DATASPACEUSER=datauser,DATASPACEDIR=/tmp SERVER=clusterhead
   RMCFG[remote_cluster] TYPE=MOAB SERVER=clusterhead:42559 DATARM=remote_data
   ```

5. Restart Moab to finalize changes. You can use the mschedctl -R command to cause Moab to automatically restart and load the changes.

   When restarting, Moab recognizes the added configuration and runs a Perl script in the Moab tool directory that configures the external scripts (also found in the tools directory) that Moab uses to perform data staging. You can view the data staging configuration by looking at the `config.dstage.pl` file in `$MOABHOMEDIR/etc`.

Advanced Configuration

If you need a more customized data staging setup, contact your account representative.

23.12.3 Peer-to-Peer SCP Key Authentication

In order to use scp as the data staging protocol, we will need to create SSH keys that allow users to copy files between the two peers, without the need for passwords. For example, if `UserA` is present on the source peer, and his counterpart is `UserB` on the destination peer, then `UserA` will need to create an SSH key and configure `UserB` to allow password-less copying. This will enable `UserA` to copy files to and from the destination peer using Moab's data staging capabilities.

Another common scenario is that several users present on the source peer are mapped to a single user on the destination peer. In this case, each user on the source peer will need to create keys and set them up with the user at the destination peer. Below are steps that can be used to set up SSH keys among two (or more) peers.

These instructions were written for OpenSSH version 3.6 and might not work correctly for older versions.

Generate SSH Key on Source Peer

As the user who will be submitting jobs on the source peer, run the following command:

```
ssh-keygen -t rsa
```

You will be prompted to give an optional key. Just hit return and ignore this or other settings. When finished, this command will create two files `id_rsa` and `id_rsa.pub` located inside the user's `~/.ssh/` directory.

Copy the Public SSH Key to the Destination Peer

Transfer the newly created public key (`id_rsa.pub`) to the destination peer:

```
scp ~/.ssh/id_rsa.pub ${DESTPEERHOST}:~
```

Disable Strict SSH Checking on Source Peer (Optional)

By appending the following to your `~/.ssh/config` file you can disable SSH prompts that ask to add new hosts to the 'known hosts file.' (These prompts can often cause problems with data staging functionality.) Note that the ${DESTPEERHOST} should be the name of the host machine running the destination peer:

```
Host ${DESTPEERHOST}
CheckHostIP no
StrictHostKeyChecking no
BatchMode yes
```

Configure Destination Peer User

Now, log in to the destination peer as the destination user and set up the newly created public key to be trusted:

```
ssh ${DESTPEERUSER}@${DESTPEERHOST}
mkdir -p .ssh; chmod 700 .ssh
cat id_rsa.pub >> .ssh/authorized_keys
chmod 600 .ssh/authorized_keys
rm id_rsa.pub
```

If multiple source users map to a single destination user, then repeat the above commands for each source user's SSH public key.

Configure SSH Daemon on Destination Peer

Some configuration of the SSH daemon may be required on the destination peer. Typically, this is done by editing the `/etc/ssh/sshd_config` file. To verify correct configuration, see that the following attributes are set (not commented):

```
---
RSAAuthentication    yes
PubkeyAuthentication yes
---
```

If configuration changes were required, the SSH daemon will need to be restarted:

```
/etc/init.d/sshd restart
```

Validate Correct SSH Configuration

If all is properly configured, if you issue the following command source peer, it should succeed without requiring a password:

```
scp ${DESTPEERHOST}:/etc/motd /tmp/
```

23.12.4 Diagnostics

Verify data staging is properly configured by using the following diagnostic commands:

- mdiag -R -v: Displays the status of the storage manager. Verify that you set up the necessary URLs.

```
> mdiag -R -v data
diagnosing resource managers
RM[data]      State: Active  Type: NATIVE  ResourceType: STORAGE
  Server:              keche
  Timeout:             30000.00 ms
  Cluster Query URL:   exec://$TOOLSDIR/grid/cluster.query.dstage.pl
  RM Initialize URL:   exec://$TOOLSDIR/grid/setup.config.pl
  System Modify URL:   exec://$TOOLSDIR/grid/system.modify.dstage.pl
  System Query URL:    exec://$TOOLSDIR/grid/system.query.dstage.pl
  Nodes Reported:      1 (scp://keche//tmp/)
  Partition:           SHARED
  Event Management:    (event interface disabled)
  Variables:           DATASPACEUSER=root,DATASPACEDIR=/tmp
  RM Languages:        NATIVE
  RM Sub-Languages:    -
```

- checknode -v: Executing this on the storage node displays the data staging operations associated with the node and its disk usage.

> The number of bytes transferred for each file is currently not used.

```
> checknode -v scp://keche//tmp/
node scp://keche//tmp/
State:      Idle  (in current state for 00:00:13)
Configured Resources: DISK: 578G
Utilized   Resources: DISK: 316G
Dedicated  Resources: ---
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Active Data Staging Operations:
  job         native.2  complete (1 bytes transferred)
(/home/brian/stage.txt)
  job         native.3  pending (1 bytes)  (/home/brian/stage.txt)
Dedicated Storage Manager Disk Usage:  0 of 592235 MB
Cluster Query URL:  exec://$TOOLSDIR/grid/cluster.query.dstage.pl
Partition:  SHARED  Rack/Slot:  ---
Flags:      rmdetected
RM[data]:   TYPE=NATIVE
EffNodeAccessPolicy: SHARED
Total Time: 00:12:15  Up: 00:12:15 (100.00%)  Active: 00:00:00 (0.00%)
Reservations:  ---
```

- mdiag -n: Displays the state of the storage node.

```
> mdiag -n
compute node summary
Name                     State   Procs      Memory         Opsys
compute1                  Idle    4:4      3006:3006       linux
compute2                  Down    0:4      3006:3006       linux
scp://keche//tmp/         Idle    0:0         0:0              -
-----                      ---    4:8      6012:6012       -----
Total Nodes: 3  (Active: 0  Idle: 2  Down: 1)
```

- checkjob -v: Displays the status of the staging request.

> The remaining time and size of the file information is currently not used. The information should only be used to see file locations and whether the file has been staged or not.

```
> checkjob -v jobid
...
Stage-In Requirements:
  localhost:/home/brian/stage.txt => keche:/tmp/staged.txt  size:0B
status:[NONE]  remaining:00:00:01
    Transfer URL: file:///home/brian/stage.txt,ssh://keche/tmp/staged.txt
...
```

To ensure that SCP key authentication is properly configured, the following conditions must be met:

- Moab is running as root.
- You are able to issue the following command as the root user without being prompted for a password:

```
su - <DATASPACEUSER> -c "/usr/bin/ssh <destination host> -l <DATASPACEUSER> 'df
-k //tmp/ 2>&1 || echo FAILED'"
```

- You can SSH `<destination host>` without a password.
- The `dataSpaceLocalUser` and `dataSpaceMappedUser` variables in your `/opt/moab/etc/config.dstage.pl` script are set to the same username you assigned through `<DATASPACEUSER>`.

23.13 Accounting and Allocation Management

In this section:

23.13.1 Peer-to-Peer Accounting

When Moab is used to manage resources across multiple clusters, there is a greater need to track and enforce the resource sharing agreements between the resource principals.

The Moab Accounting Manager is an accounting management system that provides usage tracking, charge accounting, and allocation enforcement for resource or service usage in cloud and technical computing environments. It acts like a bank in which credits are deposited into accounts with constraints designating which entities can access the account. As resources or services are utilized, accounts are charged and usage recorded. MAM supports familiar operations such as deposits, withdrawals, transfers, and refunds and provides balance and usage feedback to users, managers, and system admins. See 5.5 Accounting, Charging, and Allocation Management for more information.

MAM can be used as a real-time debiting system where jobs are charged at the moment of completion. When used in a multi-site (grid) environment, MAM facilitates trust by allowing lending organizations to manage what the costing rules are for usage of their resources and job submitters to determine how much their job will cost them before they start, ensuring all parties can agree to the transaction and giving each party a first-hand accounting record.

If the clusters are within a common administrative domain and have a common user space, then a single Moab Accounting Manager will suffice to manage the project allocation and accounting. This works best in Moab Grid Control / Moab Grid Member grids.

23.13.2 Peer-to-Peer Allocation Management

The following steps provide an example of setting up the Moab Accounting Manager to manage the allocation and accounting for a multiple cluster grid within a single administrative domain.

First you will need to install Moab Accounting Manager and its database on one or more head nodes. The following is a sample installation. See 'Installing Moab Accounting Manager' in the *Moab HPC Suite Installation and Configuration Guide* for more information.

```
# Install Prerequisites (Perl with suidperl, PostgreSQL, libxml2, ...)
[root]  yum install perl perl-suidperl postgresql postgresql-libs postgresql-devel
postgresql-server libxml2 libxml2-devel ncurses-devel readline-devel openssl
# Unpack the tarball
[root]  passwd adaptive
[adaptive]  mkdir ~/src
[adaptive]  cd ~/src
[adaptive]  gzip -cd mam-10.1.0.tar.gz | tar xvf -
[adaptive]  cd mam-10.1.0
# Install
[adaptive]  ./configure
[adaptive]  make
[root]  make deps
[root]  make install
# Configure, create and bootstrap the database
[root]  service postgresql initdb
[postgres]  echo "host    all         all         192.168.1.1    255.255.255.255
trust" >>var/lib/pgsql/data/pg_hba.conf
[postgres]  /usr/local/pgsql/bin/postmaster -i -D /usr/local/pgsql/data >var/log/pgsql
2>&1 &
[postgres]  createuser adaptive
[adaptive]  createdb mam
[adaptive]  psql mam < hpc.sql
# Startup the mam server daemon
[root]  cp contrib/init.d/mam/redhat /etc/init.d/mam
[root]  chmod +x /etc/init.d/mam
[root]  service mam start
```

Related Topics

- 23.11 Grid Credential Management

23.14 Grid Security

Secret Key Based Server Authentication

Secret key based security is required in order for the grid to work. It is enabled in the `moab-private.cfg` file. Configuration of `moab-private.cfg` is covered throughout the grid configuration documentation, and in Appendix E: Security.

23.15 Grid Diagnostics and Validation

Peer Management

- Use mdiag -R to view interface health and performance/usage statistics.
- Use mrmctl to enable/disable peer interfaces.
- Use mrmctl -m to dynamically modify/configure peer interfaces.

Peer Diagnostic

- Use *`mdiag -R`* to diagnose general resource manager interfaces.
- Use *`mdiag -S`* to diagnose general scheduler health.
- Use *`mdiag -R -V job <RMID>`* to diagnose peer-to-peer job migration:

```
> mdiag -R -V job peer1
```

- Use *`mdiag -R -V data <RMID>`* to diagnose peer-to-peer data staging.
- Use *`mdiag -R -V cred <RMID>`* to diagnose peer-to-peer credential mapping.

Chapter 24: Data Staging

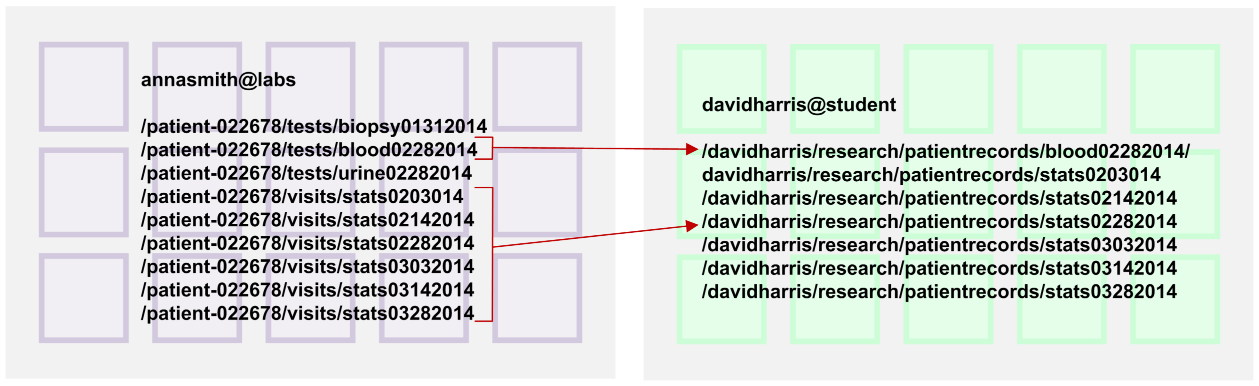
Sometimes you might need a job to process data that resides at another site. With the proper configuration, you can submit your job with the requirement that it copies data from the external site to yours and, if needed, copy the job's resulting data out to the external site for its owner to use. Data staging is an out-of-band method of moving data without reserving compute nodes or other resources for it.

In this chapter:

24.1 Data Staging Example

In the example below, which will appear throughout the chapter, a university researcher needs the results of tests done at a hospital to run his job. User `davidharris` on the `student` server of the university submits a job called `Moab.1` that requires several files stored by user `annasmith` on the `research` server of the hospital. `davidharris` submitted `Moab.1` with certain options in place that instruct Moab to copy the files to the `/student/davidharris/research/patientrecords/` directory on the `student` server prior to starting the job.

Image 24-1: Data Staging Example


annasmith@labs
/patient-022678/tests/biopsy01312014
/patient-022678/tests/blood02282014
/patient-022678/tests/urine02282014
/patient-022678/visits/stats0203014
/patient-022678/visits/stats02142014
/patient-022678/visits/stats02282014
/patient-022678/visits/stats03032014
/patient-022678/visits/stats03142014
/patient-022678/visits/stats03282014
davidharris@student
/davidharris/research/patientrecords/blood02282014/
davidharris/research/patientrecords/stats0203014
/davidharris/research/patientrecords/stats02142014
/davidharris/research/patientrecords/stats02282014
/davidharris/research/patientrecords/stats03032014
/davidharris/research/patientrecords/stats03142014
/davidharris/research/patientrecords/stats03282014

Moab currently supports the following data staging use cases: 1) Staging data to or from a shared file system, 2) Staging data to or from local node storage on a single compute node, and 3) Staging data to or from a shared file system on an unspecified cluster – resolved at job migration – in a grid configuration.

Before you can submit data staging jobs, you must configure certain generic metrics in your partitions, job templates, and the data staging submit filter for data staging scheduling, throttling, and policies.

Moab uses Linux file transfer utilities to stage the data and includes data staging reference scripts that support the scp and rsync Linux file transfer utilities. The scripts will work for standard installations, but you can customize the script to support data staging to and from an external staging server, the Moab server itself, or a local compute node, depending on your implementation. You can also customize your own script for other file transfer utilities, such as Aspera.

Once you configure your system to support data staging, you can begin creating data staging jobs by attaching the `--stagein`, `--stageinfile`, `--stageinsize`, `--stageout`, `--stageoutfile`, and `--stageoutsize` options to your *`msub`* commands. See the section Staging Data for more information.

Related Topics

- 3.7.28.H Applying the msub Submit Filter

24.2 Data Staging Tasks

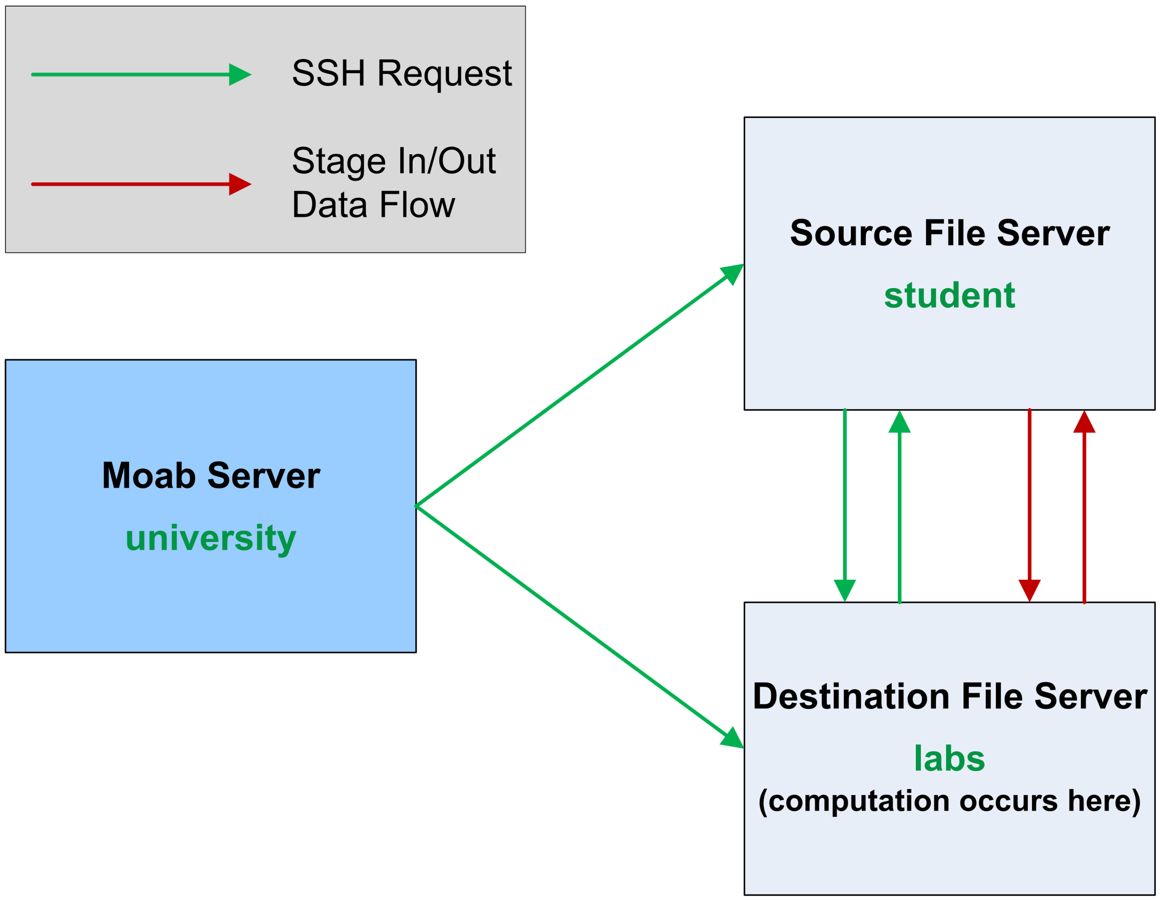
In this section:

24.2.1 Configuring the SSH Keys for the Data Staging Transfer Script

For data staging to work correctly, you must configure SSH keys to allow the data staging scripts to run without passphrases. In the sample data staging server configuration shown in the image below, `davidharris` on the `student` server stages data from the source server `student` to the destination server `labs`. The computation occurs on the `labs` server before Moab stages the output data from `labs` back to `student`.

The image below demonstrates the SSH connections necessary and how you should configure your SSH keys:

Image 24-2: Data Staging Server Configuration



For more information on generating keys, see the ssh-keygen man page and SSH login without password.

To Configure the SSH Keys for the Data Staging Transfer Script

1. Generate a new SSH key on the Moab server (`university`) if one does not already exist. To do so, run each of the following steps:
   a. Run `ssh-keygen` to generate a public and private rsa key pair:

```
davidharris@university]$ ssh-keygen
```

   b. Enter the name of the file where you want to store the key, or you can accept the default location:

```
/home/davidharris/.ssh/id_rsa
```

   c. When prompted for a passphrase, leave it blank and press `Enter`. Repeat when prompted to retype passphrase.

2. Install the public key on the source and destination hosts. Note that in this example the source host is `student` and the destination host is `labs`:

   a. Copy the `university` public key to `student`. Answer `yes` to continue connecting:

   ```
   [davidharris@university]$ ssh-copy-id -i ~/.ssh/id_rsa.pub student
   ```

   b. Copy the `university` public key to `labs`. Answer `yes` to continue connecting:

   ```
   [davidharris@university]$ ssh-copy-id -i ~/.ssh/id_rsa.pub labs
   ```

> The next two steps generate a key-pair for each node. It is acceptable to generate a single key-pair and install it on each node. It does not matter where the key-pair is generated, so long as it is compatible with the SSH client/server.

3. Generate a key pair on the source host (`student`) and install the public key generated to the destination host (`labs`). When prompted for a passphrase, leave it blank and press `Enter`. Repeat when prompted to retype passphrase.

   ```
   [davidharris@student]$ ssh-keygen
   [davidharris@student]$ ssh-copy-id -i ~/.ssh/id_rsa.pub labs
   ```

4. Generate a key pair on the destination host (`labs`) and install the public key generated to the source host (`student`). When prompted for a passphrase, leave it blank and press `Enter`. Repeat when prompted to retype passphrase.

   ```
   [davidharris@labs]$ ssh-keygen
   [davidharris@labs]$ ssh-copy-id -i ~/.ssh/id_rsa.pub student
   ```

5. Ensure that each user who will run data staging jobs has read and write permissions on each source and destination server.

6. Test the configuration:

   a. Install the modules required to run the data staging scripts. `python-paramiko` is required for data staging, but `python-mock` is only required if you intend to run the unit test:

   ```
   > yum install python-paramiko python-mock
   ```

   b. Transfer a file from the source host to the destination host to verify that the keys work for the users configured. To do so, run `/opt/moab/tools/data-staging/ds_move_scp --test=<source>%<destination>` if you use scp or `/opt/moab/tools/data-staging/ds_move_rsync --test=<source>%<destination> script` if you use rsync. `<source>%<destination>` is configured the same way as the `--stagein` and

`--stageout` options for msub; for help configuring your source and destination, see the section Staging a File or Directory.

```
[davidharris@university]$ /opt/moab/tools/data-staging/ds_move_rsync --
test=davidharris@student:/tmp/test%davidharris@labs:/tmp
```

c. In the same way, transfer a file from the destination host to the source host to verify that the keys work for the users configured:

```
[davidharris@university]$ /opt/moab/tools/data-staging/ds_move_rsync --
test=davidharris@labs:/tmp/test%davidharris@student:/test_processed
```

Related Topics

- Chapter 24: Data Staging

24.2.2 Configuring Data Staging

You must modify your Moab configuration to enable data staging. In addition to the configuration steps described below, you might also consider customizing the configuration (including the associated scripts) to meet your site's specific needs.

For advanced configuration steps and options, see 24.2.6 Configuring Data Staging with Advanced Options.

To Configure Data Staging

1. Verify that your firewall and network are correctly configured to allow the scripts to operate as designed.

2. If you have not already done so, install the modules required to run the data staging scripts. `python-paramiko` is required for data staging, but `python-mock` is only required if you intend to run the unit test.

```
> yum install python-paramiko python-mock
```

3. If you have not already, follow the instructions found in 24.2.1 Configuring the SSH Keys for the Data Staging Transfer Script.

4. Ensure that the data staging scripts are installed on your system. To do so, list the contents of the `/opt/moab/tools/data-staging` directory. You should see the data staging README file, reference scripts, and other related files.

```
> ls -l /opt/moab/tools/data-staging
```

You can copy and modify the reference scripts and configuration files to meet your specific needs. See the README file packaged in the `data-staging` directory for information about modifying these files.

5. Open your `moab.cfg` file for editing and do each of the following tasks:

   a. Configure the data staging *msub* filter, located in `/opt/moab/tools/data-staging` by default, as a client-side filter. See 3.7.28.H Applying the msub Submit Filter for more information.

   ```
   SUBMITFILTER /opt/moab/tools/data-staging/ds_filter
   ```

   The data staging filter checks the *msub* argument syntax to verify that the arguments make sense and are consistent; attempts a dry run connection via SSH and the file transfer utility to ensure that keys exist for the user on the necessary systems; and attempts to determine the size of the data that will be transferred.

   You can customize the script to meet your specific needs; the file contains detailed comments illustrating its default behavior to facilitate its modification. If you replace or modify the submit filter, it is your responsibility to ensure that the same functionality described in the paragraph above is present in your filter.

   Note that this filter has the `DEFAULT_TEMPLATE` name, which should match the name of the master data staging template in `moab.cfg`. For more information, see 24.2.6 Configuring Data Staging with Advanced Options.

   b. Set the data staging bandwidth gmetric (`DATASTAGINGBANDWIDTH_MBITS_PER_SEC`) on each partition associated with a resource manager to the rate at which its network to be used for data staging transfers data in megabits per second (see6.2.5 Per-Partition Settings for more information). Moab will use the specified rate and the data staging size specified at job submission (see the section Stage in or out File Size for more information) to determine how long staging the data will take and to schedule the job as soon after data staging completes as possible.

   Example 24-1: Non-grid

   ```
   RMCFG[torque]  Type=pbs
   PARCFG[torque] GMETRIC[DATASTAGINGBANDWIDTH_MBITS_PER_SEC]=58
   ```

   Partition `torque` has a transfer rate of 58 megabits per second. Moab uses the rate when it estimates the time it will take to stage data in and determine when to schedule the job that will use the data.

   Example 24-2: Grid

   ```
   RMCFG[m1] type=Moab
   PARCFG[m1] GMETRIC[DATASTAGINGBANDWIDTH_MBITS_PER_SEC]=100
   ```

Partition `m1` has a transfer rate of 100 megabits per second. Moab uses the rate when it estimates the time it will take to stage data in and determine when to schedule the job that will use the data.

c. Set the bandwidth generic resource on all nodes to limit the total number of concurrent data staging jobs in your system:

```
NODECFG[GLOBAL] GRES=bandwidth:10
```

Data staging jobs can use up to 10 units of bandwidth on the system. You can specify the number of units consumed by each data staging job when you configure the data staging job templates.

d. Configure moab with JOBMIGRATEPOLICY JUSTINTIME.

DataStaging requires JOBMIGRATEPOLICY JUSTINTIME to ensure the workflow job IDs are not altered upon submission.

6. Install the msub client filter on all client submission hosts.

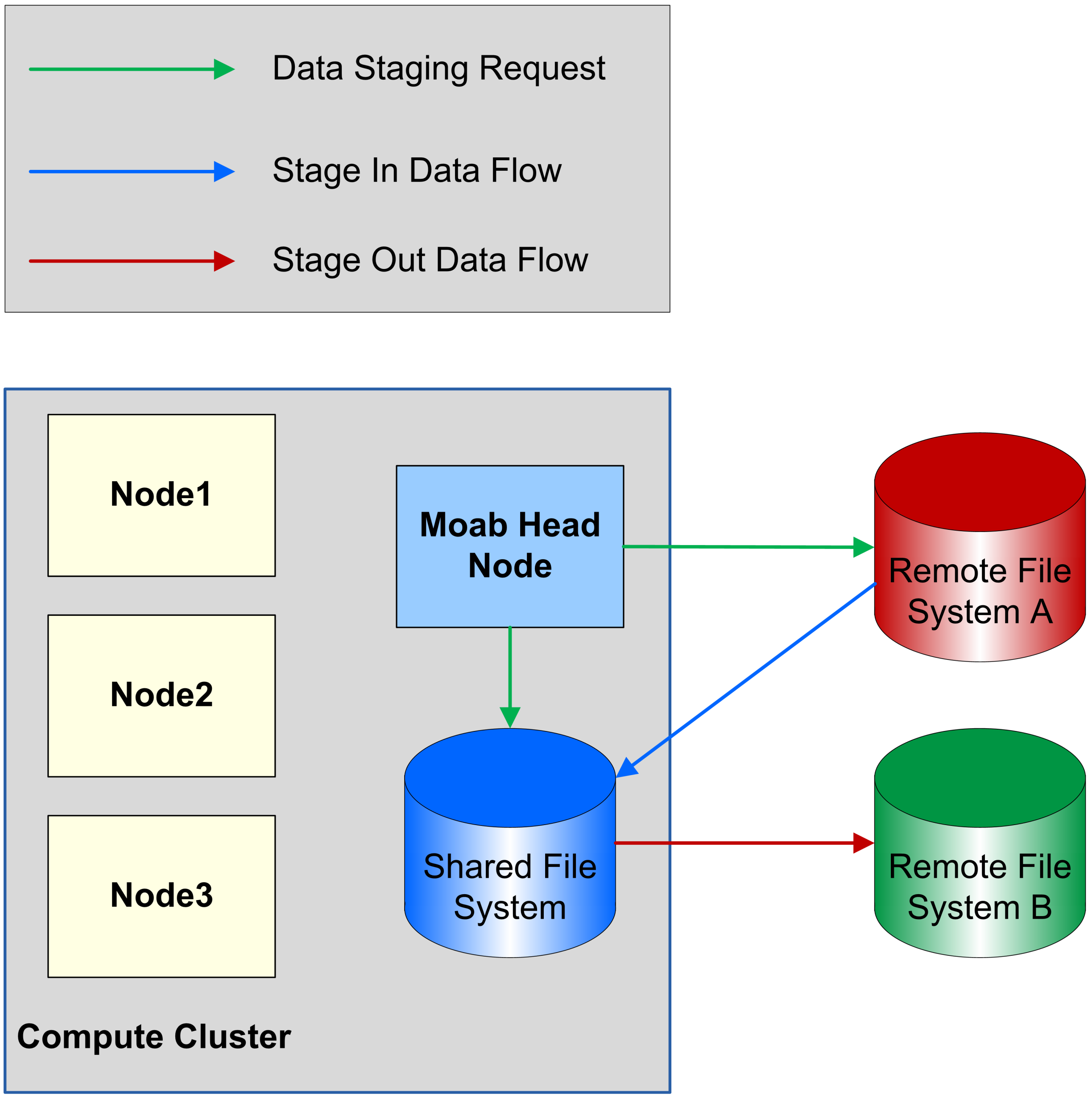
Related Topics

- Chapter 24: Data Staging

24.2.3 Staging Data to or from a Shared File System

In the most common data staging use case, the cluster utilizes a shared file system between all compute nodes. This type of data staging makes data stored outside of the cluster available to a job that will run on any set of nodes in the cluster. At the time of submission, you must specify where Moab will obtain the data with a username, host name, and path to a file or directory and where on the shared file system Moab will store the data. After the job runs, you can also copy data from the shared file system back to a remote file system.

Image 24-3: Data Staging To or From a Shared File System



To Stage Data to or from a Shared File System

1. If you have not already done so, configure your SSH keys and `moab.cfg` to support data staging. See 24.2.1 Configuring the SSH Keys for the Data Staging Transfer Script and 24.2.2 Configuring Data Staging for more information.
2. Create your job templates for data staging jobs in `moab.cfg`. The templates in the example below create a compute job that stages data in before it starts and stages data

out when it completes. For more information about creating job templates, see Chapter 22: Job Templates.

a. Create a selectable master template, called ds in the example below, that creates a stage in and stage out system job. This name should match the DEFAULT_TEMPLATE value in ds_config.py. See 24.2.6 Configuring Data Staging with Advanced Options for more information.

b. For the data staging in job template, called dsin in the example below, specify that it will create a data staging job by setting DATASTAGINGJOB to TRUE. Note that the name of this job template must match the name of the data stage in job template referenced in the master template.

c. Set the bandwidth GRES to the amount of bandwidth a single stage in job should use. This indicates how many of the bandwidth units specified with NODECFG[GLOBAL] (in 24.2.2 Configuring Data Staging), a data staging job with this template should consume.

d. Add FLAGS=GRESONLY to indicate that this data staging job does not require any compute resources.

e. Create a trigger that executes the ds_move_scp, ds_move_rsync, or ds_move_multiplex script, depending on which file transfer utility you use. Set the attacherror, objectxmlstdin, and user FLAGs to attach any trigger stderr as a message to the job, pass the job XML to the script, and indicate that the script should run as the job's user, respectively.

If you use the rsync protocol, you can configure your data staging jobs to report the actual number of bytes transferred and the total data size to be transferred. To do so, use the Sets attribute to ^BYTES_IN.^DATA_SIZE_IN for stage in jobs and ^BYTES_OUT.^DATA_SIZE_OUT for stage out jobs. For example, a stage in trigger looks like the following:

```
JOBCFG[dsin]
TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stagein",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
IN.^DATA_SIZE_IN
```

A stage out trigger looks like the following:

```
JOBCFG[dsout]
TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stageout",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
OUT.^DATA_SIZE_OUT
```

These variables show up as events if you set your WIKIEVENTS parameter to TRUE.

f. Create the stage out job, called `dsout` in the example below, by repeating steps 2b - 2e in a new template. In the example below, this template is called `dsout`. Note that the name of this job template must match the name of the data stage out job template referenced in the data staging master template.

```
JOBCFG[ds]      TEMPLATEDEPEND=AFTEROK:dsin TEMPLATEDEPEND=BEFORE:dsout
SELECT=TRUE

JOBCFG[dsin]    DATASTAGINGSYSJOB=TRUE
JOBCFG[dsin]    GRES=bandwidth:2
JOBCFG[dsin]    FLAGS=GRESONLY
JOBCFG[dsin]    TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dsout]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]   GRES=bandwidth:1
JOBCFG[dsout]   FLAGS=GRESONLY
JOBCFG[dsout]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stageout",Flags=attacherror:objectxmlstdin:user
```

3. Create the job using *`msub`*, adding resources and specifying a script as you normally would. Then configure Moab to stage the data for it:

   a. At the end of the command, use the `--stagein`/`--stageout` option and/or `--stageinfile`/`--stageoutfile` option.

   - The `--stagein`/`--stageout` option lets you specify a single file or directory to stage in or out. You must set the option equal to `<source>%<destination>`, where `<source>` and `<destination>` are both `[<user>@]<host>:/<path>/[<fileName>]`. See the section Staging a File or Directory for format and details.

     If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

     ```
     > msub --stagein=annasmith@labs:/patient-
     022678/%davidharris@university:/davidharris/research/patientrecords
     <jobScript>
     ```

     Moab copies the `/patient-022678` directory from the hospital's `labs` server to the `university` cluster where the job will run prior to job start.

   - The `--stageinfile`/`--stageoutfile` option lets you specify a file that contains the file and/or directory name(s) to stage in or out. You must set the option equal to `<path>/<fileName>` of the file. The file must contain at least one line with this format: `<source>%<destination>`, where both `<source>` and `<destination>` are `[<user>@]<host>:/<path>[<fileName>]`. See the section Staging Multiple Files or Directories for more information.

If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

```
> msub --stageinfile=/davidharris/research/recordlist <jobScript>
```

Moab copies all files specified in the `/davidharris/research/recordlist` file to the cluster where the job will run prior to job start.

`/davidharris/research/recordlist`:

```
annasmith@labs:/patient-
022678/tests/blood02282014%davidharris@university:/davidharris/research/patie
ntrecords/blood02282014
annasmith@labs:/patient-
022678/visits/stats02032014%davidharris@university:/davidharris/research/pati
entrecords/stats02032014
annasmith@labs:/patient-
022678/visits/stats02142014%davidharris@university:/davidharris/research/pati
entrecords/stats02142014
annasmith@labs:/patient-
022678/visits/stats02282014%davidharris@university:/davidharris/research/pati
entrecords/stats02282014
annasmith@labs:/patient-
022678/visits/stats03032014%davidharris@university:/davidharris/research/pati
entrecords/stats03032014
annasmith@labs:/patient-
022678/visits/stats03142014%davidharris@university:/davidharris/research/pati
entrecords/stats03142014
annasmith@labs:/patient-
022678/visits/stats03282014%davidharris@university:/davidharris/research/pati
entrecords/stats03282014
```

Moab copies the seven patient record files from the hospital's `labs` server to the `university` cluster where the job will run prior to job start.

b. The `--stageinsize`/`--stageoutsize` option lets you specify the estimated size of the files and/or directories to help Moab more quickly and accurately calculate the amount of time it will take to stage the data and therefore schedule your job correctly. If you are staging data out, then setting `--stageoutsize` is required. If you provide an integer, Moab will assume the number is in megabytes. To change the unit, add another suffix. See the section Stage in or out File Size for more information.

```
> msub --stageinfile=/davidharris/research/recordlist --stageinsize=100
<jobScript>
```

Moab copies the `/davidharris/research/recordlist` file, which is approximately 100 megabytes, from the biology node to the host where the job will run prior to job start.

4. To see the status, errors, and other details associated with your data staging job, run *`checkjob`* `-v`. See Example 3-4: Using checkjob -v on a data staging job for details.

Related Topics

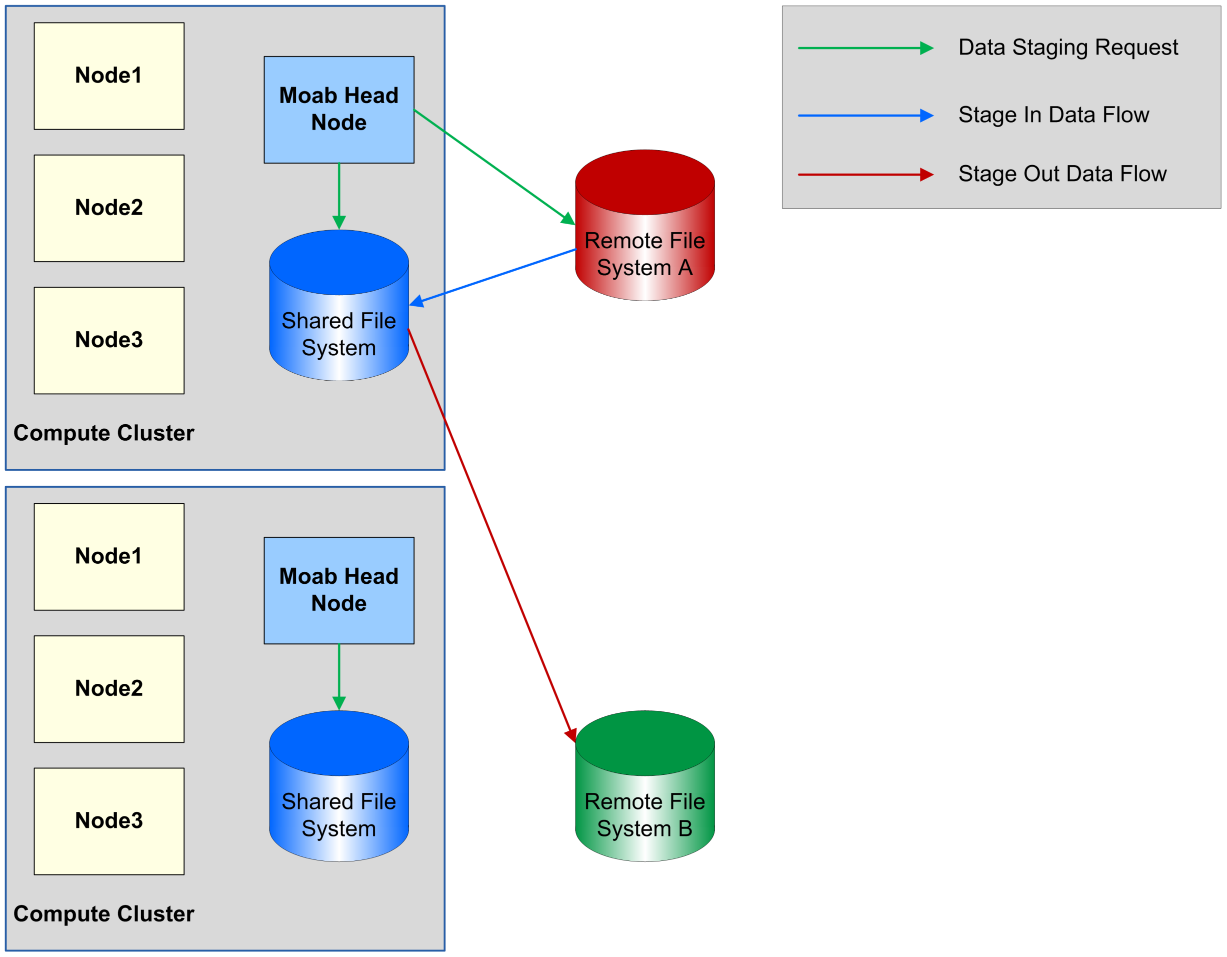
- Chapter 24: Data Staging

24.2.4 Staging Data to or from a Shared File System in a Grid

You can stage data in an environment where multiple instances of Moab run in a grid configuration. For this type of data staging, each cluster utilizes a shared file system with all compute nodes. This type of data staging will make data available to a job that will run on a set of nodes in one of the clusters in the grid. You must specify where the remote data can be obtained with a username, host name, and path to a file or directory and where on the shared storage Moab will store the data. The remote data source location is known at job submission time, but you must use the `$CLUSTERHOST` placeholder for the host name of the data transfer server on which the job will be scheduled. After the job runs, you can also copy data from the cluster shared file system to a remote file system.

Note that you cannot stage data to or from a local compute node with its own local storage in a grid environment.

Image 24-4: Data Staging in a Grid



To Stage Data to or from a Shared File System in a Grid

1. If you have not already done so, configure your SSH keys and `moab.cfg` to support data staging. See 24.2.1 Configuring the SSH Keys for the Data Staging Transfer Script and 24.2.2 Configuring Data Staging for more information.
2. Create your job templates for data staging jobs in `moab.cfg`. The templates in the example below create a compute job that stages data in before it starts and stages data out when it completes. For more information about creating job templates, see Chapter 22: Job Templates.
   a. Create a selectable master template, called `ds` in the example below, that creates a stage in and stage out system job. This name should match the `DEFAULT_TEMPLATE` value in `ds_config.py`. For more information, see 24.2.6 Configuring Data Staging with Advanced Options.

b. For the data staging in job template, called `dsin` in the example below, specify that it will create a data staging job by setting `DATASTAGINGJOB` to `TRUE`. Note that the name of this job template must match the name of the data stage in job template referenced in the master template.

c. Set the staging job template bandwidth `GRES` to the amount of bandwidth a single stage in job should use. This indicates how many of the bandwidth units specified with `NODECFG[GLOBAL]` (in 24.2.2 Configuring Data Staging), a data staging job with this template should consume.

d. Set `JOBMIGRATEPOLICY` to `JUSTINTIME`.

e. Add `FLAGS=GRESONLY` to indicate that this data staging job does not require any compute resources.

f. Create a trigger that executes the `ds_move_scp`, `ds_move_rsync`, or `ds_move_multiplex` script, depending on which file transfer utility you use. Set the `attacherror`, `objectxmlstdin`, and `user` `FLAG`s to attach any trigger stderr as a message to the job, pass the job XML to the script, and indicate that the script should run as the job's user, respectively.

If you use the rsync protocol, you can configure your data staging jobs to report the actual number of bytes transferred and the total data size to be transferred. To do so, use the Sets attribute to `^BYTES_IN.^DATA_SIZE_IN` for stage in jobs and `^BYTES_OUT.^DATA_SIZE_OUT` for stage out jobs. For example, a stage in trigger looks like the following:

```
JOBCFG[dsin]
TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stagein",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
IN.^DATA_SIZE_IN
```

A stage out trigger looks like the following:

```
JOBCFG[dsout]
TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stageout",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
OUT.^DATA_SIZE_OUT
```

These variables show up as events if you set your WIKIEVENTS parameter to `TRUE`.

g. Create the stage out job, called `dsout` in the example below, by repeating steps 2b - 2f in a new template. In the example below, this template is called `dsout`. Note that the name of this job template must match the name of the data stage out job template referenced in the master template.

```
JOBCFG[ds]      TEMPLATEDEPEND=AFTEROK:dsin TEMPLATEDEPEND=BEFORE:dsout
SELECT=TRUE
```

```
JOBCFG[dsin]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dsin]   GRES=bandwidth:2
JOBCFG[dsin]   FLAGS=GRESONLY
JOBCFG[dsin]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dsout]  DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]  GRES=bandwidth:2
JOBCFG[dsout]  FLAGS=GRESONLY
JOBCFG[dsout]  TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stageout",Flags=attacherror:objectxmlstdin:user
```

3. Create the job using *msub*, adding resources and specifying a script as you normally would. Then configure Moab to stage the data for it:

   a. At the end of the command, use the `--stagein`/`--stageout` option and/or `--stageinfile`/`--stageoutfile` option.

      - The `--stagein`/`--stageout` option lets you specify a single file or directory to stage in or out. You must set the option equal to `<source>%<destination>`, where `<source>` and `<destination>` are both `[<user>@]<host>:/<path>/[<fileName>]`. See the section Staging a File or Directory for format and details.

        Note that if you do not know the cluster where the job will run but want the data staged to the same location, you can use the `$CLUSTERHOST` variable in place of a host. If you choose to use the `$CLUSTERHOST` variable, you must first customize the `ds_config.py` file. For more information, see the section Configuring the $CLUSTERHOST Variable below.

        If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

        ```
        > msub ... --stagein=annasmith@labs:/patient-
        022678/%\$CLUSTERHOST:/davidharris/research/patientrecords <jobScript>
        ```

        Moab copies the `/patient-022678` directory from the hospital's `labs` server to the cluster where the job will run prior to job start.

      - The `--stageinfile`/`--stageoutfile` option lets you specify a file that contains the file(s) and directory(-ies) to stage in or out. You must set the option equal to `<path>/<fileName>` of the file. The file must contain at least one line with this format: `[<user>@]<host>:/<path>[<fileName>]`. See the section Staging Multiple Files or Directories for more information.

        If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

```
> msub ... --stageinfile=/davidharris/research/recordlist <jobScript>
```

Moab copies all files specified in the `/davidharris/research/recordlist` file to the cluster where the job will run prior to job start.

`/davidharris/research/recordlist`:

```
annasmith@labs:/patient-
022678/tests/blood02282014%$CLUSTERHOST:/davidharris/research/patientrecords/
blood02282014
annasmith@labs:/patient-
022678/visits/stats02032014%$CLUSTERHOST:/davidharris/research/patientrecords
/stats02032014
annasmith@labs:/patient-
022678/visits/stats02142014%$CLUSTERHOST:/davidharris/research/patientrecords
/stats02142014
annasmith@labs:/patient-
022678/visits/stats02282014%$CLUSTERHOST:/davidharris/research/patientrecords
/stats02282014
annasmith@labs:/patient-
022678/visits/stats03032014%$CLUSTERHOST:/davidharris/research/patientrecords
/stats03032014
annasmith@labs:/patient-
022678/visits/stats03142014%$CLUSTERHOST:/davidharris/research/patientrecords
/stats03142014
annasmith@labs:/patient-
022678/visits/stats03282014%$CLUSTERHOST:/davidharris/research/patientrecords
/stats03282014
```

Moab copies the seven patient record files from the hospital's `labs` server to the cluster where the job will run prior to job start.

b. The `--stageinsize`/`--stageoutsize` option lets you specify the estimated size of the files and/or directories to help Moab more quickly and accurately calculate the amount of time it will take to stage the data and therefore schedule your job correctly. If you used the `$CLUSTERHOST` variable to stage in, then setting `--stageinsize` is required. `--stageoutsize` is always required for staging data out. If you provide an integer, Moab will assume the number is in megabytes. To change the unit, add another suffix. See the section Stage in or out File Size for more information.

```
> msub ... --stageinfile=/davidharris/research/recordlist --stageinsize=100
<jobScript>
```

Moab copies the `/davidharris/research/recordlist` file, which is approximately 100 megabytes, from the biology node to the host where the job will run prior to job start.

4. To see the status, errors, and other details associated with your data staging job, run *checkjob* `-v`. See Example 3-4: Using checkjob -v on a data staging job for details.

Configuring the $CLUSTERHOST Variable

When you submit a data staging job in a grid environment, you can use the `$CLUSTERHOST` variable instead of specifying a destination if you do not know the cluster where the job will run but want the data staged to the same location. Before the variable will work correctly, you must first configure it by customizing the `ds_config.py` script to match your unique system.

Use Case

In a grid with three clusters, you have a partition named `master` where you want all data staged to a host named `gridheadNAS`; a partition named `csdept` where you want all data staged to a host named `fs001.cs.example.edu`; and a partition named `lab` where you want all data staged to a host named `bigfilesystem`.

1. Open the `ds_config.py` file for modification (located in `/opt/moab/tools/data-staging/` by default):

```
[moab]$ vi /opt/moab/tools/data-staging/ds_config.py
```

2. Locate the `PARTITION_TO_HOST` parameter:

```
...
PARTITION_TO_HOST = {"partition_1_name":"cluster_1_staging_hostname",
                     "partition_2_name":"cluster_2_staging_hostname",
                     "partition_3_name":"cluster_3_staging_hostname"}
...
```

3. Replace the partition names and associated cluster hostnames with those that match your system. For the use case provided above, you customize it this way:

```
...
PARTITION_TO_HOST = {"master":"gridheadNAS",
                     "csdept":"fs001.cs.example.edu",
                     "lab":"bigfilesystem"}
...
```

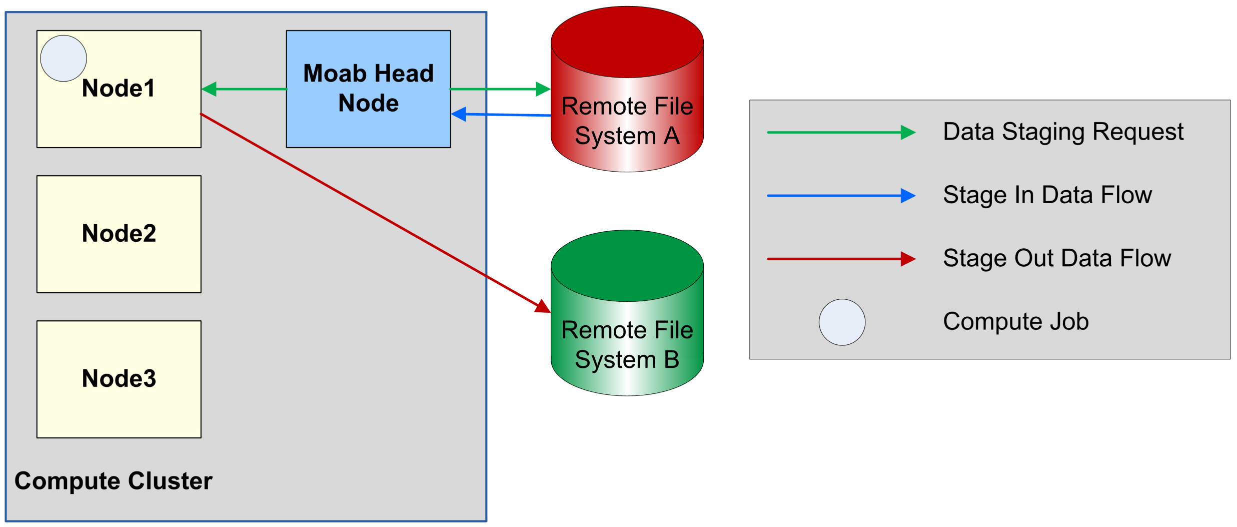
Related Topics

- Chapter 24: Data Staging

24.2.5 Staging Data To or From a Compute Node

You can stage data to or from a local compute node in an environment where each node on the cluster has local storage. This type of data staging will make data stored outside the cluster available to a job that will run on a single node in the cluster. You must specify the username, host name, and path to a file or directory and a location on the compute node where Moab will store the data. You will supply the remote data source location at job submission time, but you must use the `$JOBHOST` placeholder for the name of the compute node. After the job runs, you can also copy data from the local file system to a remote file system.

Image 24-5: Data Staging To or From a Local Compute Node



Before staging data to or from a local compute node, follow the procedure in 24.2.2 Configuring Data Staging.

To Stage Data to or from a Local Compute Node

1. If you have not already done so, configure your SSH keys and `moab.cfg` to support data staging. See 24.2.1 Configuring the SSH Keys for the Data Staging Transfer Script and 24.2.2 Configuring Data Staging for more information.
2. Create your job templates for data staging jobs in `moab.cfg`. The templates in the example below create a compute job that stages data in before it starts and stages data out when it completes. For more information about creating job templates, see Chapter 22: Job Templates.
   a. Create a selectable master template, called `ds` in the example below, that creates a stage in and stage out system job. This name should match the `DEFAULT_TEMPLATE` value in `ds_config.py`. For more information, see 24.2.6 Configuring Data Staging with Advanced Options.
   b. For the data staging in job template, called `dsin` in the example below, specify that it will create a data staging job by setting `DATASTAGINGJOB` to `TRUE`. Note that the name of this job template must match the name of the data stage in job template referenced in the master template.
   c. Set the staging job template bandwidth `GRES` to the amount of bandwidth a single stage in job should use. This indicates how many of the bandwidth units specified with `NODECFG[GLOBAL]` (in 24.2.2 Configuring Data Staging), a data staging job with this template should consume.

d. For local node data staging it is important that the data staging job has the entire node to itself. To prevent Moab from scheduling another job on the node at the same time as the data staging job, set the `NODEACCESSPOLICY` to `SINGLEJOB` in the staging job template.

e. Add `INHERITRES=TRUE` to reserve the compute node for the data staging job to prevent other compute jobs from using the node at the same time and creating input, output, and disk conflicts with the data staging job.

f. Create a trigger that executes the `ds_move_scp`, `ds_move_rsync`, or `ds_move_multiplex` script, depending on which file transfer utility you use. Set the `attacherror`, `objectxmlstdin`, and `user` `FLAG`s to attach any trigger stderr as a message to the job, pass the job XML to the script, and indicate that the script should run as the job's user, respectively.

> If you use the rsync protocol, you can configure your data staging jobs to report the actual number of bytes transferred and the total data size to be transferred. To do so, use the Sets attribute to `^BYTES_IN.^DATA_SIZE_IN` for stage in jobs and `^BYTES_OUT.^DATA_SIZE_OUT` for stage out jobs. For example, a stage in trigger looks like the following:
>
> ```
> JOBCFG[dsin]
> TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
> move_rsync --stagein",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
> IN.^DATA_SIZE_IN
> ```
>
> A stage out trigger looks like the following:
>
> ```
> JOBCFG[dsout]
> TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
> move_rsync --stageout",Flags=objectxmlstdin:user:attacherror,Sets=^BYTES_
> OUT.^DATA_SIZE_OUT
> ```
>
> These variables show up as events if you set your WIKIEVENTS parameter to `TRUE`.

g. Create the stage out job, called `dsout` in the example below, by repeating steps 2b - 2f in a new template. In the example below, this template is called `dsout`. Note that the name of this job template must match the name of the data stage out job template referenced in the data staging master template.

```
JOBCFG[ds]      TEMPLATEDEPEND=AFTEROK:dsin TEMPLATEDEPEND=BEFORE:dsout
SELECT=TRUE

JOBCFG[dsin]    DATASTAGINGSYSJOB=TRUE
JOBCFG[dsin]    GRES=bandwidth:2
JOBCFG[dsin]    NODEACCESSPOLICY=SINGLEJOB
JOBCFG[dsin]    INHERITRES=TRUE
```

```
JOBCFG[dsin]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dsout]  DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]  GRES=bandwidth:1
JOBCFG[dsout]  NODEACCESSPOLICY=SINGLEJOB
JOBCFG[dsout]  INHERITRES=TRUE
JOBCFG[dsout]  TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_rsync --stageout",Flags=attacherror:objectxmlstdin:user
```

3. Create the job using *msub*, adding resources and specifying a script as you normally would. Then configure Moab to stage the data for it:

   a. If the compute job does not use all of the node's processors, Moab could schedule another job on the node at the same time. If you did not set `NODEACCESSPOLICY` to `SINGLEJOB` in your `moab.cfg`, set the policy for this job by adding `-l naccesspolicy=singlejob` to your *msub* command.

   ```
   > msub -l naccesspolicy=singlejob... <jobScript>
   ```

   b. At the end of the command, use the `--stagein`/`--stageout` option and/or `--stageinfile`/`--stageoutfile` option.

      - The `--stagein`/`--stageout` option lets you specify a single file or directory to stage in or out. You must set the option equal to `<source>%<destination>`, where `<source>` and `<destination>` are both `[<user>@]<host>:/<path>/[<fileName>]`. See the section Staging a File or Directory for format and details.

        If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

        If you do not know the host where the job will run but want the data staged to the same location, you can use the `$JOBHOST` variable in place of a host:

        ```
        > msub --stagein=annasmith@labs:/patient-
        022678/%\$JOBHOST:/davidharris/research/patientrecords <jobScript>
        ```

        Moab copies the `/patient-022678` directory from the hospital's `labs` server to the node where the job will run prior to job start.

      - The `--stageinfile`/`--stageoutfile` option lets you specify a file that contains the file and directory name(s) to stage in or out. You must set the option equal to `<path>/<fileName>%` of the file. The file must contain at least one line with this format: `<source>%<destination>`, where `<source>` and `<destination>` are both `[<user>@]<host>:/<path>[/<fileName>]`. See the section Staging Multiple Files or Directories for more information.

If the destination partition is down or does not have configured resources, the data staging workflow submission will fail.

```
> msub --stageinfile=/davidharris/research/recordlist <jobScript>
```

Moab copies all files specified in the `/davidharris/research/recordlist` file to the host where the job will run prior to job start.

`/davidharris/research/recordlist`:

```
annasmith@labs:/patient-
022678/tests/blood02282014%$JOBHOST:/davidharris/research/patientrecords/bloo
d02282014
annasmith@labs:/patient-
022678/visits/stats02032014%$JOBHOST:/davidharris/research/patientrecords/sta
ts02032014
annasmith@labs:/patient-
022678/visits/stats02142014%$JOBHOST:/davidharris/research/patientrecords/sta
ts02142014
annasmith@labs:/patient-
022678/visits/stats02282014%$JOBHOST:/davidharris/research/patientrecords/sta
ts02282014
annasmith@labs:/patient-
022678/visits/stats03032014%$JOBHOST:/davidharris/research/patientrecords/sta
ts03032014
annasmith@labs:/patient-
022678/visits/stats03142014%$JOBHOST:/davidharris/research/patientrecords/sta
ts03142014
annasmith@labs:/patient-
022678/visits/stats03282014%$JOBHOST:/davidharris/research/patientrecords/sta
ts03282014
```

Moab copies the seven patient record files from the hospital's `labs` server to the host where the job will run prior to job start.

c. The `--stageinsize`/`--stageoutsize` option lets you specify the estimated size of the files and/or directories to help Moab more quickly and accurately calculate the amount of time it will take to stage the data and therefore schedule your job correctly. If you used the `$JOBHOST` variable to stage in, then setting `--stageinsize` is required. `--stageoutsize` is always required for staging data out. If you provide an integer, Moab will assume the number is in megabytes. To change the unit, add another suffix. See the section Stage in or out File Size for more information.

```
> msub --stageinfile=/davidharris/research/recordlist --stageinsize=100
<jobScript>
```

Moab copies the `/davidharris/research/recordlist` file, which is approximately 100 megabytes, from the biology node to the host where the job will run prior to job start.

4. To see the status, errors, and other details associated with your data staging job, run *checkjob* -v. See Example 3-4: Using checkjob -v on a data staging job for details.

   > Your checkjob output might include a warning that says "req 1 RM (internal) does not match job destination RM". You can safely ignore this message.

Related Topics

- Chapter 24: Data Staging

24.2.6 Configuring Data Staging with Advanced Options

In this topic:

24.2.6.A Using a Different Default Template Name

When you submit a data staging job, a data staging job template is attached to the job automatically. In the reference script configuration, the default template name is `ds`. This is the template that will be attached to the compute job by the client *msub* filter.

If you want to change the name of the default template that is automatically attached, you should change the value of `DEFAULT_TEMPLATE` in the `ds_config.py` file installed on all client submit hosts. This name must match the master data staging template name specified in the Moab configuration file.

To Configure the DEFAULT_TEMPLATE Variable

1. Open the `ds_config.py` file for modification (located in `/opt/moab/tools/data-staging/` by default):

```
[moab]$ vi /opt/moab/tools/data-staging/ds_config.py
```

2. Locate the `DEFAULT_TEMPLATE` parameter:

```
...
DEFAULT_TEMPLATE = "ds"
...
```

3. Replace the template name with the one specified in the Moab configuration file:

```
ds_config.py
...
DEFAULT_TEMPLATE = "datastaging"
...

moab.cfg
...
JOBCFG[datastaging] TEMPLATEDEPEND=...
```

4. Make these changes on all client submit hosts.

24.2.6.B Supporting Multiple File Transfer Script Utilities in a Grid on a Per-Partition Basis

If you want a different transfer script to run based on which partition the job is submitted to, you can configure a multiplexer script that will switch execution to various other scripts based on the partition.

To Support Multiple File Transfer Script Utilities in a Grid on a Per-Partition Basis

1. Configure the trigger in your job templates in `moab.cfg` to run `ds_move_multiplex` instead of `ds_move_rsync` or `ds_move_scp`.
2. Configure the `PARTITION_TO_SCRIPT` variable in `ds_config.py` to provide a mapping from each partition to the desired script to run.
   a. Open the `ds_config.py` file for modification (located in `/opt/moab/tools/data-staging/` by default):

   ```
   [moab]$ vi /opt/moab/tools/data-staging/ds_config.py
   ```

   b. Locate the `PARTITION_TO_SCRIPT` parameter:

   ```
   ...
   PARTITION_TO_SCRIPT =
   {"partition_1_name":"/opt/moab/tools/data-staging/ds_move_rsynch",
    "partition_2_name":"/opt/moab/tools/data-staging/ds_move_scp",
    "partition_3_name":"/opt/moab/tools/data-staging/ds_move_rsync"}
   ...
   ```

   c. Replace the `partition_*_name`s with partitions that exist in your configuration. After each partition, specify the script that you want to execute for that partition.

24.2.6.C Receiving Notification at the Completion of the Data Staging Job

If you want explicit notification in case of failure of the stage out job, add an additional trigger to the dsout job template, which will send email notification to the job's submitter. For more information, see 17.2.2 Using a Trigger to Send Email.

```
JOBCFG[dsout]  DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]  GRES=bandwidth:1
JOBCFG[dsout]  FLAGS=GRESONLY
JOBCFG[dsout]  TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stageout",Flags=attacherror:objectxmlstdin:user
JOBCFG[dsout]  TRIGGER=EType=fail,AType=mail,Action="Your (stageout) data staging job
$OID failed."
```

The first trigger listed in the template configuration should be the exec trigger. Add the email trigger and any other triggers after the exec trigger. You can modify the email trigger to run at completion rather than at failure. You can also add this type of trigger to stage in jobs.

24.2.6.D Adding a Non-Default Template via msub

You can have multiple data staging template workflows defined in the `moab.cfg`. The submit filter is configured to add only one of them by default. If you want to use one of the other available templates, you can do so by using the `-l template=TEMPLATENAME` option in the *`msub`* command.

Given the following `moab.cfg`:

```
#Default data staging template:

JOBCFG[ds]     TEMPLATEDEPEND=AFTEROK:dsin TEMPLATEDEPEND=BEFORE:dsout SELECT=TRUE
JOBCFG[dsin]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dsin]   GRES=bandwidth:2
JOBCFG[dsin]   FLAGS=GRESONLY
JOBCFG[dsin]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dsout]  DATASTAGINGSYSJOB=TRUE
JOBCFG[dsout]  GRES=bandwidth:1
JOBCFG[dsout]  FLAGS=GRESONLY
JOBCFG[dsout]  TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-staging/ds_
move_rsync --stageout",Flags=attacherror:objectxmlstdin:user

#experimental data staging template:

JOBCFG[dscustom]     TEMPLATEDEPEND=AFTEROK:dscustomin
TEMPLATEDEPEND=BEFORE:dscustomout SELECT=TRUE
JOBCFG[dscustomin]   DATASTAGINGSYSJOB=TRUE
JOBCFG[dscustomin]   GRES=bandwidth:2
JOBCFG[dscustomin]   FLAGS=GRESONLY
JOBCFG[dscustomin]   TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_custom --stagein",Flags=attacherror:objectxmlstdin:user

JOBCFG[dscustomout]  DATASTAGINGSYSJOB=TRUE
JOBCFG[dscustomout]  GRES=bandwidth:1
```

```
JOBCFG[dscustomout]  FLAGS=GRESONLY
JOBCFG[dscustomout]  TRIGGER=EType=start,AType=exec,Action="/opt/moab/tools/data-
staging/ds_move_custom --stageout",Flags=attacherror:objectxmlstdin:user
```

The user could submit a job using the custom data staging template with the following command:

```
[moab]$ msub -l template=dscustom …
```

24.2.6.E Using msub to Return all the Job IDs in the Workflow at Submission Time

By default, *msub* will print the job ID to stdout at the time of submission. If you want to have *msub* print all of the jobs that are created as part of the data staging workflow template, you can use the *msub* `--workflowjobids` option to show all the job IDs at submission time:

```
$ echo sleep 60 | msub -l walltime=15 --workflowjobids

MoabA.3.dsin MoabA.3 MoabA.3.dsout
```

This could be useful if you are writing scripts to do your own workflows and you need to programmatically capture the data stage out job name for use in your workflow.

24.3 Data Staging References

Sample User Job Script

The code below is an example of a job script that a user might use to run a data staging job:

```
#!/bin/bash
#
# Sample data staging job script
#
# stage in directives
#MSUB --stageinsize=1MB
#MSUB --stagein=davidharris@source-server:/tmp/filein.tostage%davidharris@destination-
server:/tmp/filein.staged
#
# stage out directives
#MSUB --stageoutsize=10GB
#MSUB --stageout=davidharris@destination-
server:/tmp/fileout.tostage%davidharris@source-server:/tmp/fileout.staged

# run executable on the destination host using staged data
$HOME/bin/my_compute_executable < /tmp/filein.staged > /tmp/fileout.tostage
```

Related Topics

- Chapter 24: Data Staging

Chapter 25: Using NUMA with Moab

Moab works with Torque to support these two Non-Uniform Memory Architecture (NUMA) systems:

- NUMA-Aware – This configuration supports multi-req jobs and jobs that span hosts.
- NUMA-Support – This configuration supports only a single instance for pbs_mom that is treated as if there were multiple nodes running in the cluster. This configuration is only for large-scale SLES systems using SGI Altix and UV hardware.

This chapter serves as a central information repository for the various configuration settings involved when using either NUMA system configuration.

Both systems cannot be configured for use at the same time.

In this chapter:

25.1 Using NUMA-Aware with Moab

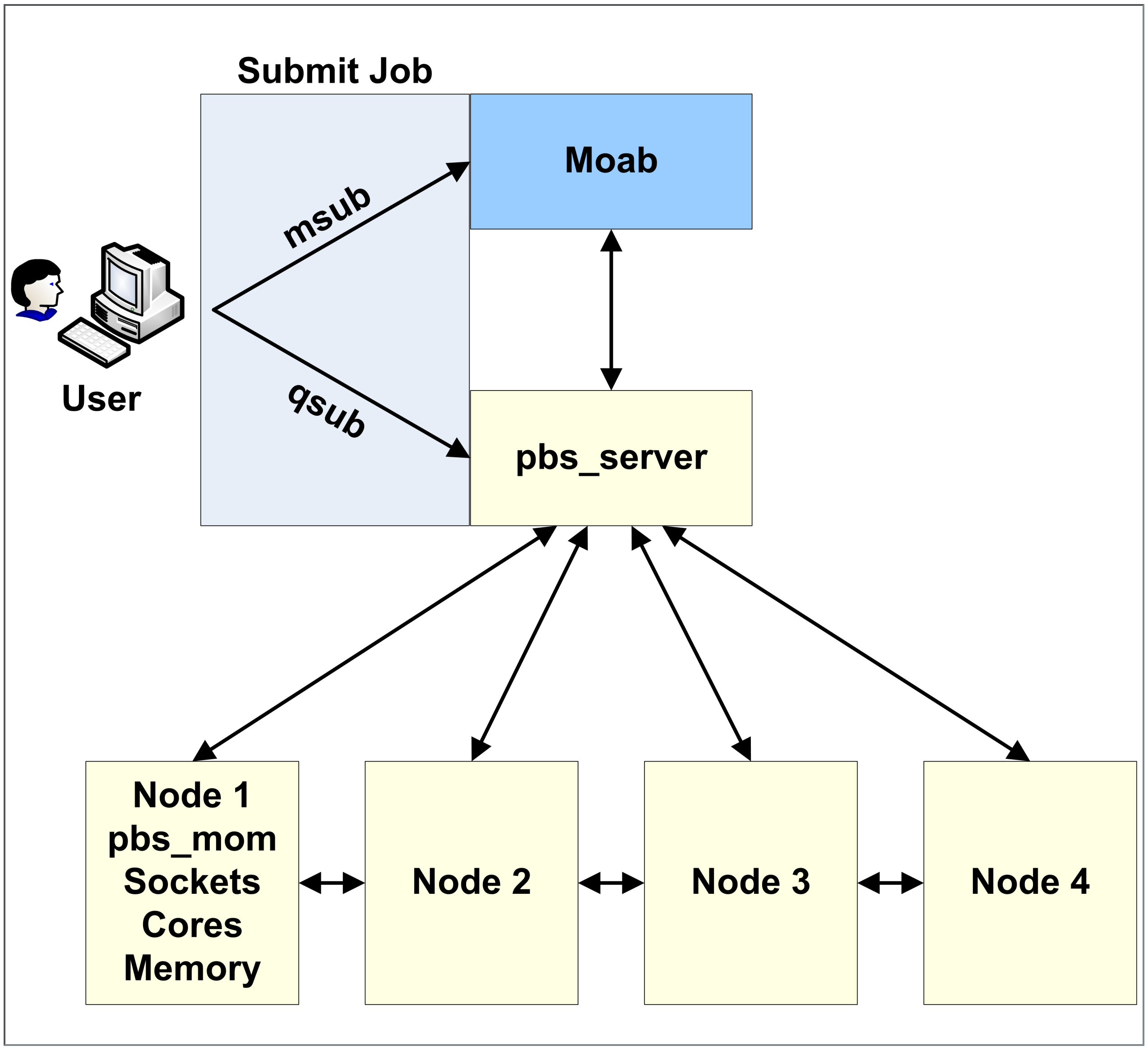
This chapter introduces NUMA-aware scheduling and identifies Moab-, Torque-, and MAM-related functions.

NUMA-aware scheduling requires Torque Resource Manager 6.0 or later.

In this section:

25.1.1 NUMA Process

The following image provides an example of the NUMA process:



25.1.2 Installation and Configuration

Moab does not require any special installation or configuration processes to support NUMA-aware with Torque.

25.1.3 Moab and NUMA Resources

Moab uses generic resources to natively understand the concept of 'socket', 'numanode', 'core' and 'thread'. The msub/qsub '-L' syntax for job submissions lets you request placement or allocation of these specific resources.

See '-L NUMA Resource Request' in the *Torque Resource Manager Administrator Guide* for more information on using the -L syntax.

For the -L syntax, submit using qsub unless your system requires submissions using msub.

- A job requesting a placement of a numanode is requesting exclusive access to the entire numa node and all of its resources including cores, threads, memory, gpus, and mics.
- A job requesting a placement of a socket is requesting exclusive access to the entire socket including numanodes, cores, threads, memory, gpus and mics.

Moab does not require a configuration change to support NUMA-aware scheduling (no new Moab configuration parameters). However, you might need to increase MAXGRES to accommodate the additional resources. See the parameter MAXGRES.

When using NUMA-aware, the following occurs:

1. Moab imports NUMA resources from Torque and treats them as a special case of generic resources.
2. When a job requests a NUMA resource, such as a socket or numanode, Moab will schedule exclusive access to those resources for the job. If exclusive access to the NUMA resource is not available on a particular node, Moab will look for another node or schedule the job out into the future if no resources are available at the time.

Balanced Resources

Moab assumes that the NUMA resources on a particular node are balanced. This means that each socket has the same amount of resources, including numanodes, cores, threads, memory, and gpus.

Moab supports advanced multi-req resource requests within the same job using the msub/qsub '-L' syntax.

In addition, any job that requests NUMA resources will receive a per-task default memory requirement if a memory requirement is *not* specified by the user.

Job Recommendations

Adaptive Computing provides these recommendations for jobs:

- Jobs that request NUMA resources can share nodes with non-NUMA jobs, but it is *not* recommended. We recommend that you enforce the separation using policies (queues, reservations, partitions, node-sets, etc.).

- GPU jobs that request cores and processors should *not* share nodes with non-GPU jobs. For example, a job requesting a numanode can be given a numanode with a GPU and therefore block other jobs from consuming that GPU. This can be prevented with proper queue policies and placement.
- A job can run across heterogeneous NUMA resources. For example, a socket on one node can contain more memory than a socket on another node. Use the NODEALLOCATIONPOLICY Moab parameter to enable running a job across homogenous NUMA resources.

25.1.4 Track Dedicated NUMA Resources

If Moab Accounting Manager is part of your configuration, you can configure MAM to track dedicated NUMA resources (sockets, numanodes, cores, threads).

As the MAM Admin, run the commands for the individual resources you want to track. The following example shows the commands for all of the available resources:

```
[mam]$ mam-shell Attribute Create Object=UsageRecord Name=Sockets DataType=Integer
Description="\"Number of Sockets Dedicated\""
[mam]$ mam-shell Attribute Create Object=UsageRecord Name=NumaNodes DataType=Integer
Description="\"Number of Numa Nodes Dedicated\""
[mam]$ mam-shell Attribute Create Object=UsageRecord Name=Cores DataType=Integer
Description="\"Number of Cores Dedicated\""
[mam]$ mam-shell Attribute Create Object=UsageRecord Name=Threads DataType=Integer
Description="\"Number of Threads Dedicated\""
```

NUMA resources are only reported to MAM when they are dedicated to the job. As you can specify the placement rules (the NUMA resources that are dedicated), it is *not* recommended to charge for any NUMA resources; use Processors instead.

25.2 Using NUMA-Support with Moab

This section serves as a central information repository for NUMA-support systems. This section provides basic information and contains links to the various NUMA-aware topics found throughout the documentation.

Support for NUMA-support systems is available only on large-scale SLES systems using SGI Altix and UV hardware and requires Torque 3.0 or later.

Installation and Configuration

Additional information is provided on configuring Moab for NUMA-support. See G.2 Hardware Integration.

Appendix A: Moab Parameters

See 2.1 Initial Moab Configuration for more information about specifying parameters.

If a parameter does not have a set default, the Default value in the table is shown as '---'.

Index: A B C D E F G H I J K L M N O P Q R S T U V W X

ACCOUNTCFG[<ACCOUNTID>]	
Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: General Credential Flags, `CHARGERATE`, PRIORITY, ENABLEPROFILING, MEMBERULIST, `PLIST`, `QDEF`, `QLIST`, usage limit, or a fairness usage limit specification (FSCAP, FSTARGET, and FSWEIGHT).
Default	---
Description	Specifies account specific attributes. See 2.7.4 Account (or Project) Credential for general information and 2.8 Job Flags for a description of valid flag values.
Example	`ACCOUNTCFG[projectX] MAXJOB=50 QDEF=highprio` *Up to `50` jobs submitted under the account ID `projectX` will be allowed to execute simultaneously and will be assigned the QOS `highprio` by default.*

ACCOUNTINGINTERFACEURL	
Format	`<URL>` Where protocol can be one of `exec` or `file`
Default	---
Description	The interface to use for real-time export of Moab accounting/auditing information. See 12.2.4.B Exporting Events in Real-Time for more information.
Example	`ACCOUNTINGINTERFACEURL exec:///$TOOLSDIR/dumpacc.pl`

ACCOUNTWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The priority weight to be applied to the specified account priority. See 4.1.2.B Credential (CRED) Component.
Example	`ACCOUNTWEIGHT 100`

ADMIN1, ADMIN2, ADMIN3	
Description	These parameters are deprecated. Use ADMINCFG.

ADMINCFG[X]	
Format	One or more <ATTR>=<VALUE> pairs. Where <ATTR> is one of the following: ENABLEPROXY, USERS, GROUPS, SERVICES, or NAME
Default	---
Description	Enables you to configure which services and users belong to a particular level of administration. **Note:** The first user listed in the ADMINCFG[1] users list is considered to be the primary admin. The option USERS=ALL is allowed. The groups list adds the groups' users as if they were listed individually as USERS. To prevent Moab from assigning a primary user from the first group listed, you must specify a primary user first using the USERS attribute, then list the desired groups.

Example	`ADMINCFG[1] USERS=root,john`
	`ADMINCFG[1] GROUPS=admin`
	`ADMINCFG[1] SERVICES=ALL`
	`ADMINCFG[1] NAME=batchadmin`
	`ADMINCFG[3] USERS=bob,carol,smoore`
	`ADMINCFG[3] GROUPS=science,math`
	`ADMINCFG[3] SERVICES=mjobctl,mcredctl,runjob`
	`ADMINCFG[3] NAME=helpdesk`
	Members of the `batchadmin` admin role and members of the `admin` group are allowed to run all commands. Members of the `helpdesk` role and `science` and `math` groups are allowed to run `mjobctl`. They are also able to view and modify credential objects (i.e., users, groups, accounts, etc.). See Appendix E: Security for more information.
	`ADMINCFG[4] USERS=ALL SERVICES=checknode`
	All users can execute checknode to get information on any node.

AGGREGATENODEACTIONS	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Consolidates queued node actions into as few actions as possible to reduce communication burden with resource manager. Node actions are queued until the parameter AGGREGATENODEACTIONSTIME setting. This might delay some node actions. When reprovisioning, the system job might expire before the provision action occurs; while the action will still occur, the job will not show it.
Example	`AGGREGATENODEACTIONS TRUE` *Queues node actions together when possible.*

AGGREGATENODEACTIONSTIME	
Format	`<SECONDS>`
Default	`60`
Description	The delay time for the AGGREGATENODEACTIONS parameter to aggregate requests before sending job batches.
Example	`AGGREGATENODEACTIONSTIME 120` *Sets the `AGGREGATENODEACTIONS` delay to two minutes.*

ALLOWMULTIREQNODEUSE	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	By default, Moab does not allow different requirements on the same job to occupy the same node. For example, if a job is submitted with `nodes=2:ppn=8+4:fast:ppn=16`, it's possible that some of the tasks requested could overlap onto the same node. This parameter instructs Moab to allow overlapping the same node, or not. This parameter also applies to the various `-w` clauses of an mshow -a command.
Example	`ALLOWMULTIREQNODEUSE TRUE`

ALLOWROOTJOBS	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether batch jobs from the root user (UID=0) are allowed to be executed. **Note**: The resource manager must also support root jobs.
Example	`ALLOWROOTJOBS TRUE` *Jobs from the root user can execute.*

ALWAYSEVALUATEALLJOBS	
Format	`ALWAYS`, `FIRSTRSV`, or `FULLRSV`
Default	`FIRSTRSV`
Description	Instructs Moab how to handle the scheduling of eligible jobs during the first phase of each scheduling iteration: • FIRSTRSV directs Moab to stop considering eligible jobs once a single reservation has been created. • FULLRSV tells Moab to evaluate eligible jobs until reservations have been created for a number of eligible jobs. • ALWAYS directs Moab to always evaluate all eligible jobs.
Example	`ALWAYSEVALUATEALLJOBS FIRSTRSV`

AMCFG	
Format	One or more key-value pairs as described in 5.6 AMCFG Parameters and Flags.
Default	---
Description	The interface and policy configuration for the scheduler-accounting manager interface.
Example	`AMCFG[mam] TYPE=MAM STARTFAILUREACTION=HOLD`

APPLICATIONLIST	
Format	Space-delimited list of generic resources.
Default	---
Description	Specifies which generic resources represent actual applications on the cluster/grid. See 10.7 Managing Consumable Generic Resources for more information.

Example	`NODECFG[node01] GRES=calclab:1,powerhouse:1 RCSOFTWARE=calclab:1,powerhouse:1`
	`NODECFG[node02] GRES=calclab:1,powerhouse:1 RCSOFTWARE=calclab:1,powerhouse:1`
	`APPLICATIONLIST calclab,powerhouse`
	The generic resources `calclab` and `powerhouse` will now be recognized and treated as application software.

ARRAYJOBPARLOCK	
Format	<BOOLEAN>
Default	FALSE
Description	If set to TRUE, all subjobs of an array are locked to a single partition. The default behavior when scheduling array subjobs is to span the jobs across partitions when possible. The ARRAYJOBPARLOCK job flag can be used to specify partition locking at submit time. The ARRAYJOBPARSPAN job flag overrides this parameter.
Example	`ARRAYJOBPARLOCK TRUE`

ATTRATTRWEIGHT	
Format	<INTEGER>
Default	0
Description	The priority weight to be applied to jobs with the specified job attribute. See 4.1.2.H Job Attribute (ATTR) Component.
Example	`ATTRATTRWEIGHT 100`

ATTRGRESWEIGHT	
Format	<INTEGER>
Default	0
Description	The priority weight to be applied to jobs requesting the specified generic resource. See 4.1.2.H Job Attribute (ATTR) Component.
Example	`ATTRGRESWEIGHT 200`

ATTRSTATEWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The priority weight to be applied to jobs with the specified job state. See 4.1.2.H Job Attribute (ATTR) Component.
Example	`ATTRSTATEWEIGHT 200`

ATTRWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The priority component weight to be applied to the ATTR subcomponents. See 4.1.2.H Job Attribute (ATTR) Component.

Example	`ATTRWEIGHT      2`
	`ATTRSTATEWEIGHT 200`

BACKFILLDEPTH	
Format	`<INTEGER>`
Default	`0` (no limit)
Description	The number of idle jobs to evaluate for backfill. The backfill algorithm will evaluate the top `<X>` priority jobs for scheduling. By default, all jobs are evaluated.
Example	`BACKFILLDEPTH 128` *Evaluate only the top `128` highest priority idle jobs for consideration for backfill.*

BACKFILLMETRIC	
Format	One of the following: PROCS, PROCSECONDS, SECONDS, or NODES.
Default	PROCS
Description	The criteria used by the backfill algorithm to determine the 'best' jobs to backfill. Only applicable when using the BESTFIT backfill algorithm.
Example	`BACKFILLMETRIC PROCSECONDS`

BACKFILLPOLICY	
Format	One of FIRSTFIT, BESTFIT, or NONE.
Default	FIRSTFIT
Description	Specifies which backfill algorithm will be used. See 7.2.2 Backfill Algorithms for more information.
Example	`BACKFILLPOLICY  NONE`

BFCHUNKDURATION	
Format	[[[DD:]HH:]MM:]SS
Default	0 (chunking disabled)
Description	The duration during which freed resources will be aggregated for use by larger jobs. Used in conjunction with BFCHUNKSIZE. See 7.2.2 Backfill Algorithms for more information.

Example	`BFCHUNKDURATION 00:05:00`
	`BFCHUNKSIZE     4`
	Aggregate backfillable resources for up to 5 minutes, making resources available only to jobs of size 4 or larger.

BFCHUNKSIZE

Format	`<INTEGER>`
Default	`0` (chunking disabled)
Description	The minimum job size that can utilize chunked resources. Used in conjunction with BFCHUNKDURATION. See 7.2.2 Backfill Algorithms for more information.

Example	`BFCHUNKDURATION 00:05:00`
	`BFCHUNKSIZE     4`
	Aggregate backfillable resources for up to `5 minutes`, making resources available only to jobs of size `4` or larger.

BFMINVIRTUALWALLTIME

Format	`[[[DD:]HH:]MM:]SS`
Default	---
Description	The minimum job wallclock time for virtual scaling (optimistic-like backfilling). Any job with a wallclock time less than this setting will *not* be virtually scaled. The value specified relates to a job's original walltime and not its virtually-scaled walltime.
Example	`BFMINVIRTUALWALLTIME  00:01:30`

BFPRIORITYPOLICY

Format	One of `RANDOM`, `DURATION`, or `HWDURATION`.
Default	---
Description	Specifies policy to use when prioritizing backfill jobs for preemption.
Example	`BFPRIORITYPOLICY  DURATION` *Use length of job in determining which backfill job to preempt.*

BFVIRTUALWALLTIMECONFLICTPOLICY

Format	PREEMPT
Default	---
Description	Specifies how to handle scheduling conflicts when a virtually scaled job 'expands' to its original wallclock time. This occurs when the job is within one scheduling iteration - RMPOLLINTERVAL - of its virtually scaled wallclock time expiring.
Example	`BFVIRTUALWALLTIMECONFLICTPOLICY   PREEMPT`

BFVIRTUALWALLTIMESCALINGFACTOR

Format	<DOUBLE>
Default	0 (virtual scaling disabled)
Description	The factor by which eligible jobs' wallclock time is virtually scaled (optimistic-like backfilling). If you do not want scaling, set BFVIRTUALWALLTIMESCALINGFACTOR to '0' (default). Setting to '1' is not recommended as it impacts performance. When set to '1', Moab will exercise the code paths of scaling but no actual scaling will occur.
Example	`BFVIRTUALWALLTIMESCALINGFACTOR .4`

BYPASSCAP

Format	<INTEGER>
Default	0
Description	The max weighted value allowed from the bypass count subfactor when determining a job's priority (see 4.1.2 Job Priority Factors for more information).

Example	`BYPASSWEIGHT  5000`
	`BYPASSCAP     30000`

BYPASSWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight to be applied to a job's backfill bypass count when determining a job's priority (see 4.1.2 Job Priority Factors for more information).
Example	`BYPASSWEIGHT 5000`

CHECKPOINTDIR	
Format	`<STRING>`
Default	---
Description	The directory for temporary job checkpoint files (usually of the form `jobid.cp`). **Note:** This is *not* the directory for Moab's checkpoint file (`.moab.ck`).
Example	`CHECKPOINTDIR  /tmp/moabcheckpoint`

CHECKPOINTEXPIRATIONTIME	
Format	`[[[DD:]HH:]MM:]SS` or `UNLIMITED`
Default	`3,000,000 seconds`
Description	Specifies how 'stale' checkpoint data can be before it is ignored and purged.
Example	`CHECKPOINTEXPIRATIONTIME 1:00:00:00` *Expire checkpoint data that has been stale for over* `1 day`.

CHECKPOINTFILE	
Format	`<STRING>`
Default	`.moab.ck`
Description	Name (absolute or relative) of the Moab checkpoint file.
Example	`CHECKPOINTFILE /var/adm/moab/.moab.ck` *Maintain the Moab checkpoint file in the file specified.*

CHECKPOINTINTERVAL	
Format	`[[[DD:]HH:]MM:]SS`
Default	`00:05:00`
Description	Time between automatic Moab checkpoints. If the parameter RMPOLLINTERVAL does not specify both a minimum and maximum poll time, Moab will ignore `CHECKPOINTINTERVAL` and checkpoint every iteration.
Example	`CHECKPOINTINTERVAL 00:15:00` *Moab should checkpoint state information every `15 minutes`.*

CHECKSUSPENDEDJOBPRIORITY	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Prevents Moab from starting a job on any node containing a suspended job of higher priority.
Example	`CHECKSUSPENDEDJOBPRIORITY    FALSE`

CHILDSTDERRCHECK	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, child processes Moab executes are considered failed if their standard error stream contains the text 'ERROR'.
Example	`CHILDSTDERRCHECK    TRUE`

CLASSCFG[<CLASSID>]	
Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: General Credential Flags, DEFAULT.ATTR, DEFAULT.DISK, DEFAULT.FEATURES, DEFAULT.GRES, DEFAULT.MEM, DEFAULT.NODE, DEFAULT.NODESET, DEFAULT.PROC, ENABLEPROFILING, EXCL.FEATURES, EXCLUDEUSERLIST, HOSTLIST, IGNHOSTLIST, JOBEPILOG, JOBPROLOG, JOBTRIGGER, MAXPROCPERNODE, MAX.NODE, MAX.PROC, MAX.TPN, MAX.WCLIMIT, MIN.NODE, MIN.PROC, MIN.TPN, MIN.WCLIMIT, PARTITION, PRIORITY, PRIORITYF, QDEF, QLIST, REQ.FEATURES, REQUIREDACCOUNTLIST, REQUIREDUSERLIST, RESFAILPOLICY, SYSPRIO, WCOVERRUN, usage limit, or fairshare usage limit specification.
Default	---
Description	Specifies class specific attributes (see 2.7.5 Class (or Queue) Credential for details).
Example	`CLASSCFG[batch] MAXJOB=50 QDEF=highprio` *Up to `50 jobs` submitted to the class `batch` will be allowed to execute simultaneously and will be assigned the QOS `highprio` by default.*

CLASSWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The weight to be applied to the class priority of each job (see 4.1.2.B Credential (CRED) Component and 2.7.1.A Credential Priority Settings/Priority Weight Offsets).
Example	`CLASSWEIGHT 10`

CLIENTCFG[<X>]	
Format	One or more of `<ATTR>-<VALUE>` pairs. Where `<X>` indicates the specified peer and `<ATTR>` is one of the following: `AUTH`, `AUTHCMD`, `AUTHTYPE`, `HOST`, `KEY`, or `DEFAULTSUBMITPARTITION`.
Default	---
Description	The shared secret key and authentication method that Moab will use to communicate with the named peer daemon. See E.1 Authentication (Interface Security) for more information. **Note**: The `AUTHTYPE` and `KEY` attributes of this parameter can only be specified in the `moab-private.cfg` config file.
Example	`CLIENTCFG[silverB] KEY=apple7 AUTH=admin1` *Moab will use the session key `apple7` for peer authentication and for encrypting and decrypting messages sent from `silverB`. Also, client connections from this interface will be authorized at an `admin 1` level.*

CLIENTCONNECTIONTIMEOUT	
Format	`<SECONDS>`
Default	`30`
Description	Specifies how long client commands will wait for the initial connection to succeed before giving up and failing.
Example	`CLIENTCONNECTIONTIMEOUT 1` *Client commands will wait only 1 second for the initial connection. If the client command has not connected within 1 second, it will give up and fail.*

CLIENTMAXCONNECTIONS	
Format	`<INTEGER>`
Default	`128`
Description	Changes the maximum number of connections that can simultaneously connect to Moab. The value can be increased during runtime, but it cannot be decreased. The value cannot be reduced below the default value of `128`.
Example	`CLIENTMAXCONNECTIONS 256` *Doubles the maximum number of connections.*

CLIENTMAXPRIMARYRETRY	
Format	`<INTEGER>` or `INFINITY`
Default	`1`
Description	Specifies how many times the client command will attempt to retry its connection to the primary server if Moab is not available.

Example	`CLIENTMAXPRIMARYRETRY 5`
	`CLIENTMAXPRIMARYRETRYTIMEOUT 1000`
	The client command will attempt to retry its connection to the primary server `5` times with `1 second` intervals before giving up. ***Note****: If `INFINITY` is specified, Moab will attempt 2,140,000,000 times.*

CLIENTMAXPRIMARYRETRYTIMEOUT	
Format	`<INTEGER>` (milliseconds)
Default	`2000`
Description	Specifies how much time to wait until the client command will attempt to retry its connection to the primary server if Moab is not available.

Example	`CLIENTMAXPRIMARYRETRY 3`
	`CLIENTMAXPRIMARYRETRYTIMEOUT 500`
	The client command will attempt to retry its connection to the primary server `3` times with `.5 second` intervals before giving up.

CLIENTTIMEOUT	
Format	`[[[DD:]HH:]MM:]SS`
Default	`00:00:30`
Description	Time that Moab client commands will wait for a response from the Moab server. See Client Interface for more information. **Note**: Can also be specified as an environment variable.
Example	`CLIENTTIMEOUT 00:15:00` *Moab clients will wait up to* `15 minutes` *for a response from the server before timing out.*

CLIENTUIPORT	
Format	`<INTEGER>`
Default	N/A
Description	Port on which to listen when *`UIMANAGEMENTPOLICY FORK`* is specified. This is typically Moab's configured listen port + 1. Both `CLIENTUIPORT` and `UIMANAGEMENTPOLICY` need to be defined on clients for them to use the backup port when the primary Moab process is busy. `UIMANAGEMENTPOLICY` should be configured on the server and any client machines.

Example	`UIMANAGEMENTPOILCY FORK`
	`CLIENTUIPORT 42560`
	Moab is typically configured to listen on port 42559.

CREDDISCOVERY	
Format	`TRUE`
Default	`FALSE`
Description	Specifies that Moab should create otherwise unknown credentials when it discovers them in the statistics files.
Example	`CREDDISCOVERY TRUE`

CREDWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The credential component weight associated with the credential priority. See 4.1.2.B Credential (CRED) Component for more information.
Example	`CREDWEIGHT 2`

DATASTAGEHOLDTYPE	
Format	Any Job Hold type.
Default	`DEFER`
Description	Specifies what to do if a job's data staging operations fail.
Example	`DATASTAGEHOLDTYPE  BATCH`

DEADLINEPOLICY

Format	One of `CANCEL`, `HOLD`, `IGNORE`, or `RETRY`.
Default	`NONE`
Description	Specifies what to do when a requested deadline cannot be reached (see 9.9 Job Deadlines).
Example	`DEADLINEPOLICY IGNORE`

DEFAULTCLASSLIST

Format	Space-delimited list of one or more `<STRING>`s.
Default	---
Description	The default classes supported on each node for resource manager systems that do not provide this information.
Example	`DEFAULTCLASSLIST serial parallel`

DEFAULTSTARTTIMEQUERY

Format	Space-delimited list of one or more `<STRING>`s.
Default	---
Description	Specifies the default classes supported on each node for resource manager systems that do not provide this information.
Example	`DEFAULTSTARTTIMEQUERY RESERVATION` *Moab will estimate job start times using reservations.*

DEFAULTSUBMITPARTITION	
Format	See the parameter CLIENTCFG[].
Default	---
Description	If a user submits a job using msub that does not specify host, feature, or partition constraints, then the *msub* client will insert the specified default submit partition into the newly submitted job as a hard requirement.
Example	`CLIENTCFG[DEFAULT] DEFAULTSUBMITPARTITION=partition1`

DEFERCOUNT	
Format	`<INTEGER>`
Default	`24`
Description	The number of times a job can be deferred before it will be placed in batch hold.
Example	`DEFERCOUNT 12`

DEFERSTARTCOUNT	
Format	`<INTEGER>`
Default	`1`
Description	The number of times a job will be allowed to fail in its start attempts before being deferred. `JOBRETRYTIME` overrides `DEFERSTARTCOUNT`; `DEFERSTARTCOUNT` only begins when the `JOBRETRYTIME` window elapses. **Note**: A job's startcount will increase each time a start request is made to the resource manager regardless of whether or not this request succeeded. This means start count increases if job starts fail or if jobs are started and then rejected by the resource manager. For related information, see 6.1.4 Reservation Policies, and the parameters DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.
Example	`DEFERSTARTCOUNT 3`

DEFERTIME

Format	`[[[DD:]HH:]MM:]SS`
Default	`1:00:00`
Description	The amount of time a job will be held in the deferred state before being released back to the Idle job queue. **Note**: A job's defer time will be restarted if Moab is restarted. For related information, see 6.1.4 Reservation Policies, and the parameters DEFERSTARTCOUNT, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.
Example	`DEFERTIME 0:05:00`

DELETESTAGEOUTFILES

Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether the scheduler should delete explicitly specified stageout files after they are successfully staged. By default, such files are not deleted but are left on the nodes where the job ran.

Example	`DELETESTAGEOUTFILES TRUE`
	`Example of an explicit stageout request`
	`msub x=MSTAGEOUT:ssh://source_node/tmp/file,file:///results_folder`
	`job.cmd`
	
	With this parameter set to `TRUE`*,* `/tmp/file` *on* `source_node` *is deleted after it is copied to the specified destination (* `file:///results_folder`*). If the parameter is not set, or if it is set to* `FALSE`*,* `/tmp/file` *remains on* `source_node` *after the job terminates.*

DEPENDFAILUREPOLICY	
Format	`HOLD` or `CANCEL`
Default	`HOLD`
Description	Specifies what happens to a job if its dependencies cannot be fulfilled; that is, what happens when a job depends on another job to complete successfully but the other job fails.
Example	`DEPENDFAILUREPOLICY CANCEL` *If job A is submitted with depend=afterok:B and job B fails, job A is canceled.*

DIRECTORYSERVER	
Format	`<HOST>[:<PORT>]`
Default	---
Description	The interface for the directory server.
Example	`DIRECTORYSERVER calli3.icluster.org:4702`

DISABLEEXCHLIST	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	If the resource manager rejects a job and the value is set to `TRUE`, then the node is not added to the job's exclude host list.
Example	`DISABLEEXCHLIST TRUE`

DISABLEINTERACTIVEJOBS	
Format	<BOOLEAN>
Default	FALSE
Description	Disallows interactive jobs submitted with msub -I. **Note**: It is possible for users to submit interactive jobs directly to a resource manager, which can bypass the DISABLEINTERACTIVEJOBS parameter. However, some resource managers (such as Torque) will check with Moab before allowing an interactive job.
Example	`DISABLEINTERACTIVEJOBS TRUE`

DISABLEREQUIREDGRESNONE	
Format	<BOOLEAN>
Default	FALSE
Description	When set to TRUE, this causes Moab to reject msub requests that have a gres of 'none'. ENFORCEGRESSACCESSS must also be set to TRUE for this feature to work.
Example	

```
##########  moab.cfg  ##########
ENFORCEGRESACCESS TRUE
DISABLEREQUIREDGRESNONE TRUE
################################

> msub -A ee -l nodes=1,ttc=5,walltime=600,partition=g02 -l gres=none
ERROR: cannot submit job - cannot locate required resource 'none'
```

DISABLESAMECREDPREEMPTION	
Format	Comma-delimited list of one or more credentials: ACCT, CLASS, GROUP, QOS, or USER.
Default	---
Description	This parameter prevents specified credentials from preempting its own jobs.
Example	`DISABLESAMECREDPREEMPTION  QOS,USER`

DISABLESCHEDULING

Format	<BOOLEAN>
Default	FALSE
Description	Specifies whether or not the scheduler will schedule jobs. If set to TRUE, Moab will continue to update node and job state but will not start, preempt, or otherwise modify jobs. The command mschedctl -r will clear this parameter and resume normal scheduling.
Example	`DISABLESCHEDULING FALSE`

DISABLESLAVEJOBSUBMIT

Format	<BOOLEAN>
Default	TRUE
Description	This parameter can be added to the moab.cfg file on an MGM Moab server (in a grid configuration) to prevent users from submitting jobs to the MGC Moab server from the MGM Moab server. Some grid configurations allow the user to submit jobs on the MGM that are migrated to the MGC and submitted from the MGC. Other grid configurations do not allow the jobs to be migrated to the MGC from the MGM, in which case, jobs submitted from the MGM remain idle on the MGM and never run. This parameter will reject the job submissions on the MGM to prevent the submission of jobs that will never run.

Example	`DISABLESLAVEJOBSUBMIT TRUE`
	`example (node04 is a slave and node06 is the master)`
	`[test@node04 moab-slave]$ echo sleep 100 | msub`
	`ERROR:    cannot submit job from slave`

DISABLETHRESHOLDTRIGGERS

Format	<BOOLEAN>
Default	FALSE
Description	This makes Moab not fire threshold-based triggers, but will log the intended action to the event logs.
Example	`DISABLETHRESHOLDTRIGGERS TRUE`

DISKWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The priority weight to be applied to the amount of dedicated disk space required per task by a job (in MB).
Example	`RESWEIGHT  10` `DISKWEIGHT 100` *If a job requires 12 tasks and 512 MB per task of dedicated local disk space, Moab will increase the job's priority by* `10` *** `100` ** 12 * 512.*

DISPLAYFLAGS

Format	One or more of the following values (space delimited): `ACCOUNTCENTRIC`, `HIDEBLOCKED`, `HIDECREDS`, `HIDEDRAINED`, `NODECENTRIC`, `USEBLOCKING`, `USENOBLOCKMSUB`
Default	---
Description	By default, no flags (special modifications) are specified. If flags are specified, this controls how Moab client commands display varied information: • `ACCOUNTCENTRIC`: Displays account information in showq, rather than group information. • `HIDEBLOCKED`: Prevents showq from listing information about blocked jobs that are not owned by the user if the user is not an admin. • `HIDECREDS`: Users without Moab administrative privileges will not be able to see the credentials of other jobs. • `HIDEDRAINED`: Prevents mdiag -n from displaying nodes in the DRAINED state. An override option of `mdiag -n -w nodestate=drained` lists only those nodes with a DRAINED state. • `NODECENTRIC`: Displays node allocation information instead of processor allocation information in showq. • `USEBLOCKING`: Disables threading for commands that support it; those commands include showq, mdiag -n, and mdiag -j. • `USENOBLOCKMSUB`: Moab will skip error checking of the msub job submission and queue it up for later processing. The job ID will be returned immediately.
Example	`DISPLAYFLAGS NODECENTRIC`

DISPLAYPROXYUSERASUSER

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, Moab shows the proxy users instead of the real user on some queries of system jobs that have proxy users. Commands affected include mjobctl -q diag and checkjob.
Example	`DISPLAYPROXYUSERASUSER TRUE`

DONTCANCELINTERACTIVEHJOBS

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, Moab does not cancel interactive jobs that are held.
Example	`DONTCANCELINTERACTIVEHJOBS TRUE`

DONTENFORCEPEERJOBLIMITS

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, only the scheduler that is running the job can cancel the job or enforce other limits.
Example	`DONTENFORCEPEERJOBLIMITS TRUE`

ENABLEFAILUREFORPURGEDJOB

Format	`<BOOLEAN>`
Default	`FALSE`
Description	By default, when a job is purged or removed by the Torque resource manager for a walltime violation, the job takes on a state of Completed and a completion code of 0. If `TRUE`, the job state is set to Removed and has a completion code of 98. `ENABLEFAILUREFORPURGEDJOB` is for the Torque resource manager only. For `ENABLEFAILUREFORPURGEDJOB` to return Removed job states, you must reset the TORQUE server attribute `keep_completed` to `0` in `qmgr`. See 'Queue Attributes' in the *Torque Resource Manager Administrator Guide* for more information.
Example	`ENABLEFAILUREFORPURGEDJOB TRUE` *Jobs that are purged or removed by Torque are given a state of Removed and a completion code of 98.*

ENABLEFSVIOLATIONPREEMPTION	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, Moab will allow jobs within the same class/queue to preempt when the preemptee is violating a fairshare target and the preemptor is not.
Example	`ENABLEFSVIOLATIONPREEMPTION TRUE`

ENABLEHIGHTHROUGHPUT	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Reduces iteration times by eliminating string error checking during checkpointing, eliminating automatic rack processing, reducing object caching, using vfork rather than fork, reducing resource manager timeout parameters, and scheduling similar jobs as a chunk rather than individually. If `ENABLEHIGHTHROUGHPUT` is `TRUE`, you must set NODEALLOCATIONPOLICY to `FIRSTAVAILABLE`.

ENABLEJOBARRAYS	
Format	`<BOOLEAN>`
Default	`TRUE`
Description	If set to `TRUE`, job arrays will be enabled.
Example	`ENABLEJOBARRAYS TRUE`

ENABLENEGJOBPRIORITY

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, the scheduler allows job priority value to range from -INFINITY to MMAX_PRIO; otherwise, job priority values are given a lower bound of '1'. For more information, see the parameter REJECTNEGPRIOJOBS.
Example	`ENABLENEGJOBPRIORITY TRUE` *Job priority can range from -INFINITY to MMAX_PRIO.*

ENABLENODEADDRLOOKUP

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, the scheduler will use the default host name service lookup mechanism (i.e., `/etc/hosts`, DNS, NIS, etc.) to determine the IP address of the nodes reported by the resource manager. This information is used to correlate information reported by multi-homed hosts.
Example	`ENABLENODEADDRLOOKUP TRUE`

ENABLEPOSUSERPRIORITY

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, the scheduler will allow users to specify positive job priority values that will be honored. In other words, users can specify a priority that falls in the range of -1024 to +1023, inclusive. If set to `FALSE` (the default), user priority values are given an upper bound of '0' when users request a positive priority. See the parameter USERPRIOWEIGHT.
Example	`ENABLEPOSUSERPRIORITY TRUE` *Users can now specify positive job priorities and have them take effect (e.g., `msub -p 100 job.cmd`).*

ENABLESPVIOLATIONPREEMPTION	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	If set to `TRUE`, Moab will allow jobs within the same class/queue to preempt when the preemptee is violating a *soft* usage policy and the preemptor is not.
Example	`ENABLESPVIOLATIONPREEMPTION TRUE`

ENFORCEACCOUNTACCESS	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether or not Moab will enforce account access constraints without an accounting manager.
Example	`ENFORCEACCOUNTACCESS TRUE`

ENFORCEGRESACCESS	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	If a user submits a job with a non-existent gres (e.g., in the case of a typo) and `ENFORCEGREACCESS` is not set in `moab.cfg`, or is set to `FALSE`, then the requested gres will be created (but will not exist on any nodes) and the job will be deferred (similar to `ENFORCEACCOUNTACCESS`).
Example	`ENFORCEGRESACCESS TRUE`

EVENTFILEFORMAT	
Format	One of `JSON`, `WHITESPACE`, or `WIKI`
Default	WIKI
Description	The format to write the event log.
Example	`EVENTFILEFORMAT WIKI`

EVENTSERVER	
Format	`<HOST>[:<PORT>]`
Default	---
Description	The interface for the event server.
Example	`EVENTSERVER  calli3.icluster.org:4702`

FEATURENODETYPEHEADER	
Format	`<STRING>`
Default	---
Description	The header used to specify node type via node features (for example, PBS node attributes).
Example	`FEATURENODETYPEHEADER xnt` *Moab will interpret all node features with the leading string `xnt` as a nodetype specification, as used by the accounting manager and other accounting managers, and assign the associated value to the node (for example, xntFast).*

FEATUREPARTITIONHEADER	
Format	`<STRING>`
Default	---
Description	The header used to specify node partition via node features (for example, PBS node attributes).
Example	`FEATUREPARTITIONHEADER xpt` *Moab will interpret all node features with the leading string* `xpt` *as a partition specification and assign the associated value to the node (for example, xptFast).*

FEATUREPROCSPEEDHEADER	
Format	`<STRING>`
Default	---
Description	The header used to extract node processor speed via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specify that only features with a trailing number be interpreted. For example, the header 'sp$' will match 'sp450' but not 'sport'.
Example	`FEATUREPROCSPEEDHEADER xps` *Moab will interpret all node features with the leading string* `xps` *as a processor speed specification and assign the associated value to the node (i.e., xps950).*

FEATURERACKHEADER	
Format	`<STRING>`
Default	---
Description	The header used to extract node rack index via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specify that only features with a trailing number be interpreted. For example, the header 'rack$' will match 'rack4' but not 'racket'.
Example	`FEATURERACKHEADER rack` *Moab will interpret all node features with the leading string rack as a `rack` index specification and assign the associated value to the node (i.e., rack16).*

FEATURESLOTHEADER	
Format	`<STRING>`
Default	---
Description	The header used to extract node slot index via node features (i.e., LL features or PBS node attributes). **Note**: Adding a trailing '$' character will specify that only features with a trailing number be interpreted. For example, the header 'slot$' will match 'slot12' but not 'slotted'.
Example	`FEATURESLOTHEADER slot` *Moab will interpret all node features with the leading string `slot` as a slot index specification and assign the associated value to the node (i.e., slot16).*

FEEDBACKPROGRAM

Format	`<STRING>`
Default	---
Description	The name of the program to be run at the completion of each job. If not fully qualified, Moab will attempt to locate this program in the 'tools' subdirectory. For more details on how this works and what fields are provided, see 18.1 User Feedback Facility.
Example	`FEEDBACKPROGRAM /var/moab/fb.pl` *Moab will run the specified program at the completion of each job.*

FILEREQUESTISJOBCENTRIC

Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether a job's file request is a total request for the job or a per task request.
Example	`FILEREQUESTISJOBCENTRIC TRUE` *Moab will treat file requests as a total request per job.*

FILTERCMDFILE

Format	`<BOOLEAN>`
Default	`TRUE`
Description	Running the msub command performs the following operations on the submission script: • Replace all comments with spaces (excludes Resource Manager directives) • Strip empty lines • Replace `\r` with `\n` • Lock job to a PBS resource manager if `$PBS` is found in the script Include the `FILTERCMDFILE` parameter in the Moab configuration file that resides on the clients (`moab.cfg` or `moab-client.cfg`). `FILTERCMDFILE` must be `FALSE` for REJECTDOSSCRIPTS to work correctly.
Example	`FILTERCMDFILE FALSE` *Running the `msub` command does not perform the actions detailed earlier.*

FORCENODEREPROVISION

Format	`<BOOLEAN>`
Default	`FALSE`
Description	When set to `TRUE`, this config option causes Moab to reprovision a node, even if it is to the same operating system (in essence rewriting the OS).
Example	`FORCENODEREPROVISION TRUE`

FORCERSVSUBTYPE	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies that admin reservations must have a subtype associated with them.
Example	`FORCERSVSUBTYPE TRUE` *Moab will require all admin reservations to include a subtype.*

FREETIMELOOKAHEADDURATION	
Format	`[[[DD:]HH:]MM:]SS`
Default	`2 Months`
Description	Specifies how far ahead Moab will look when calculating free time on a node.
Example	`FREETIMELOOKAHEADDURATION 7:00:00:00` *Moab will look `1 week` ahead when it calculates free time on a node.*

FSACCOUNTWEIGHT	
Format	`<INTEGER>`
Default	`1000`
Description	The weight assigned to the account subcomponent of the fairshare component of priority.
Example	`FSACCOUNTWEIGHT 10`

FSCAP	
Format	<DOUBLE>
Default	0 (NO CAP)
Description	The maximum allowed absolute value for a job's total preweighted fairshare component.
Example	`FSCAP 10.0` *Moab will bind a job's preweighted fairshare component by the range +/- 10.0.*

FSCLASSWEIGHT	
Format	<INTEGER>
Default	1000
Description	The weight assigned to the class subcomponent of the fairshare component of priority.
Example	`FSCLASSWEIGHT 10`

FSDECAY	
Format	<DOUBLE>
Default	1.0
Description	Specifies decay rate applied to past fairshare interval when computing effective fairshare usage. Values can be in the range of 0.01 to 1.0. A smaller value causes more rapid decay causing *aged* usage to contribute less to the overall effective fairshare usage. A value of 1.0 indicates that no decay will occur and all fairshare intervals will be weighted equally when determining effective fairshare usage. See 5.3 Fairshare.

Example	`FSPOLICY   DEDICATEDPS`
	`FSDECAY    0.8`
	`FSDEPTH    8`
	Moab will apply a decay rate of 0.8 to all fairshare windows.

FSDEPTH	
Format	<INTEGER>
Default	8
Description	**Note**: The number of available fairshare windows is bounded by the MAX_ FSDEPTH value (32 in Moab). See 5.3 Fairshare.
Example	`FSDEPTH 12`

FSDISABLEIFNEGATIVE	
Format	<BOOLEAN>
Default	FALSE
Description	Disable the fairshare component of priority if it is negative for a particular job.
Example	`FSDISABLEIFNEGATIVE TRUE` *Moab will disable any fairshare components that are negative when determining a job's priority.*

FSENABLECAPPRIORITY	
Format	<BOOLEAN>
Default	FALSE
Description	Fairshare priority will increase to target and stop.
Example	`FSENABLECAPPRIORITY TRUE`

FSGROUPWEIGHT	
Format	`<INTEGER>`
Default	`1000`
Description	The weight assigned to the group subcomponent of the fairshare component of priority.
Example	`FSGROUPWEIGHT 4`

FSINTERVAL	
Format	`[[[DD:]HH:]MM:]SS`
Default	`12:00:00`
Description	The length of each fairshare window.
Example	`FSINTERVAL 12:00:00` *Track fairshare usage in `12 hour` blocks.*

FSJPUWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The fairshare weight assigned to jobs per user.
Example	`FSJPUWEIGHT 10`

FSMOSTSPECIFICLIMIT	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	When checking policy usage limits in a fairshare tree, if the most specific policy limit is passed then do not check the same policy again at higher levels in the tree. For example, if a user has a MaxProc policy limit then do not check the MaxProc policy limit on the account for this same user.
Example	`FSMOSTSPECIFICLIMIT  TRUE`

FSPOLICY	
Format	`<POLICY>[*]` See 5.3.1.A FSPOLICY - Specifying the Metric of Consumption for valid values.
Default	`[NONE]`
Description	The unit of tracking fairshare usage.
Example	`FSPOLICY DEDICATEDPES` *Moab will track fairshare usage by dedicated processor-equivalent seconds.*

FSPPUWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The fairshare weight assigned to processors per user.
Example	`FSPPUWEIGHT 10`

FSPSPUWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The fairshare weight assigned to processor-seconds per user.
Example	`FSPSPUWEIGHT 10`

FSQOSWEIGHT	
Format	`<INTEGER>`
Default	`1000`
Description	The priority weight assigned to the QOS fairshare subcomponent.
Example	`FSQOSWEIGHT 16`

FSTARGETISABSOLUTE	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether Moab should base fairshare targets off of delivered cycles or up/available cycles.
Example	`FSTARGETISABSOLUTE TRUE`

FSTREE	
Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: SHARES or MEMBERLIST
Default	---
Description	The share tree distribution for job fairshare prioritization (see 5.3.3 Hierarchical Fairshare/Share Trees).
Example	`FSTREE[geo]  SHARES=16  MEMBERLIST=geo103,geo313,geo422`

FSTREEACLPOLICY	
Format	`OFF`, `PARENT`, or `FULL`
Default	`FULL`
Description	Specifies how Moab should interpret credential membership when building the FSTREE (see 5.3.3 Hierarchical Fairshare/Share Trees).
Example	`FSTREEACLPOLICY PARENT` *Credentials will be given access to their parent node when applicable.*

FSTREEISREQUIRED	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether a job must have an applicable node in the partition's FSTREE in order to execute within that partition (see 5.3.3 Hierarchical Fairshare/Share Trees).
Example	`FSTREEISREQUIRED TRUE` *Jobs must have an applicable node in the FSTREE in order to execute.*

FSTREEUSERISREQUIRED	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether the user must be given explicit access to a branch in the FSTREE (see 5.3.3 Hierarchical Fairshare/Share Trees).
Example	`FSTREEUSERISREQUIRED TRUE` *Users must be given explicit access to FSTREE nodes in order to gain access to the FSTREE.*

FSUSERWEIGHT	
Format	`<INTEGER>`
Default	`1000`
Description	The priority weight assigned to the user fairshare subfactor.
Example	`FSUSERWEIGHT 8`

FSWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The priority weight assigned to the summation of the fairshare subfactors (see 4.1.2 Job Priority Factors and 5.3 Fairshare).
Example	`FSWEIGHT 500`

GEVENTCFG[<GEVENT>]	
Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. See 10.9.1 Configuring Generic Events for details on values you can assign to each attribute.
Default	---
Description	Specifies how the scheduler should behave when various cluster events are detected. See 10.9 Enabling Generic Events for more information.

Example	`GEVENTCFG[hitemp] ACTION=avoid,record,notify  REARM=00:10:00`
	`GEVENT[nodeerror] SEVERITY=3`
	If a `hitemp` *event is detected, Moab adjusts the node allocation policy to minimize the allocation of the node. Moab also sends emails to cluster administrators and reports the event in the Moab event log.*

GRESCFG[<GRES>]	
Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: `FEATUREGRES`, `LICENSE`, `INVERTTASKCOUNT`, `PRIVATE`, `TYPE`, or `STARTDELAY`.
Default	---
Description	Specifies associations of generic resources into resource groups: • When `LICENSE` is set to `TRUE`, Moab will pass this generic resource to the accounting manager in the Licenses property rather than the Resources property. • When `PRIVATE` is set to `TRUE`, Moab puts the requested generic resource on a separate job request. By default, a private request is a request with 1 task with X number of generic resources per task. • When `INVERTTASKCOUNT` and `PRIVATE` are set to `TRUE`, Moab makes the generic resource's private request a request with `X` number of tasks with 1 generic resource per task. See 12.6 Managing Consumable Generic Resources for more information.

Example	`GRESCFG[scsi1] TYPE=fastio`
	`GRESCFG[scsi2] TYPE=fastio`
	`GRESCFG[scsi3] TYPE=fastio`
	The generic resources `scsi1`, `scsi2`, and `scsi3` are all associated with the generic resource type `fastio`.

GRESTOJOBATTR	
Format	Comma-delimited list of generic resources.
Default	---
Description	The list of generic resources will also be interpreted as JOB features. See 6.1.5 Configuring and Managing Reservations.
Example	`GRESTOJOBATTR  matlab,ccs` *Jobs that request the generic resources `matlab` or `ccs` will have a corresponding job attribute assigned to them.*

GROUPCFG[<GROUPID>]	
Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: General Credential Flags, `PRIORITY`, ENABLEPROFILING, `QLIST`, `QDEF`, `PLIST`, `FLAGS`, usage limits, or a fairshare usage limit specification.
Default	---
Description	Specifies group specific attributes. See 2.8 Job Flags for a description of valid flag values.
Example	`GROUPCFG[staff] MAXJOB=50 QDEF=highprio` *Up to 50 jobs submitted by members of the group `staff` will be allowed to execute simultaneously and will be assigned the QOS `highprio` by default.*

GROUPWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The priority weight assigned to the specified group priority (see 4.1.2.B Credential (CRED) Component).
Example	`GROUPWEIGHT 20`

GUARANTEEDPREEMPTION	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Causes Moab to lock PREEMPTOR jobs until JOBRETRYTIME expires (essentially, waiting for the PREEMPTEE jobs to finish). It may take some time for the PREEMPTEE jobs to clear out. During that time, the PREEMPTOR job might want to look elsewhere to run, which would be disruptive as it might preempt another set of jobs. If you want to prevent this, we recommend that you set `GUARANTEEDPREEMPTION` to `TRUE`. For related information, see Chapter 21: Preemption, 6.1.4 Reservation Policies, and the parameters DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, and JOBRETRYTIME.
Example	`GUARANTEEDPREEMPTION TRUE`

HALOCKCHECKTIME	
Format	[[[DD:]HH:]MM:]SS
Default	9
Description	Specifies how frequently the secondary server checks the timestamp on the lock file. See 18.2 Enabling High Availability Features for more information.
Example	`HALOCKCHECKTIME 00:00:15` *The Moab fallback server will check the health of the Moab primary server every 15 seconds.*

HALOCKUPDATETIME	
Format	[[[DD:]HH:]MM:]SS
Default	3
Description	Specifies how frequently the primary server checks the timestamp on the lock file. See 18.2 Enabling High Availability Features for more information.
Example	`HALOCKUPDATETIME 00:00:03` *The Moab primary server will check the timestamp of the lock file every 3 seconds.*

IDCFG	
Format	One or more of the following attribute/value pairs: `BLOCKEDCREDLIST`, `CREATECRED`, `REFRESHPERIOD`, `REQUIREUSERLIST`, `RESETCREDLIST`, or `SERVER`. See 18.4 Identity Managers for additional information.
Default	---
Description	This parameter enables the identity manager interface allowing credential, policy, and usage information to be shared with an external information service. Only one identity manager can be configured at a time.
Example	`IDCFG[info] SERVER=exec:///usr/local/bin/dbquery.pl REFRESHPERIOD=30:00` *Moab will refresh credential info every half hour using the STDOUT of the specified script.*

IGNOREMDATASTAGING	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	When set to `TRUE`, Moab will ignore any resource manager specific data staging on a job and assume the resource manager is processing the request. Currently, this only applies to PBS.
Example	`IGNORERMDATASTAGING TRUE`

IGNORECLASSES	
Format	`[!]<CLASS>[,<CLASS>]...`
Default	---
Description	By default, if using the Torque resource manager, jobs from all listed classes are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not** (i.e., '!') character is specified, only jobs from listed classes are processed. See 14.1.4 Moab Side-by-Side for more information.
Example	`IGNORECLASSES dque,batch` *Moab will ignore jobs from classes `dque` and `batch`.*

IGNOREJOBS	
Format	`[!]<JOBID>[,<JOBID>]...`
Default	---
Description	By default, listed jobs are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not** (i.e., '!') character is specified, only listed jobs are processed. See 14.1.4 Moab Side-by-Side for more information.
Example	`IGNOREJOBS !14221,14223` *Moab will ignore jobs all classes except `14221` and `14223`.*

IGNORENODES

Format	`[!]<NODE>[,<NODE>]...`
Default	---
Description	By default, all listed nodes are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not** (i.e., '!') character is specified, only listed nodes are processed. See 14.1.4 Moab Side-by-Side for more information.
Example	`IGNORENODES !host3,host4` *Moab will only process nodes* `host3` *and* `host4`*.*

IGNOREPREEMPTEEPRIORITY

Format	`<BOOLEAN>`
Default	`FALSE`
Description	By default, preemptor jobs can only preempt preemptee jobs if the preemptor has a higher job priority than the preemptee. When this parameter is set to true, the priority constraint is removed allowing any preemptor to preempt any preemptees once it reaches the top of the eligible job queue.
Example	`IGNOREPREEMPTEEPRIORITY TRUE` *A preemptor job can preempt any preemptee jobs when it is at the top of the eligible job queue.*

IGNOREUSERS

Format	`[!]<USERNAME>[,<USERNAME>]...`
Default	---
Description	By default, if using the Torque resource manager, jobs from all listed users are ignored and not scheduled, tracked, or otherwise processed by Moab. If the **not** (i.e., '!') character is specified, only jobs from listed users are processed. See 14.1.4 Moab Side-by-Side for more information.
Example	`IGNOREUSERS testuser1,annapolis` *Moab will ignore jobs from users `testuser1` and `annapolis`.*

#INCLUDE

Format	`<STRING>`
Default	---
Description	Specifies another file that contains more configuration parameters. If `<STRING>` is not an absolute path, Moab will search its home directory for the file.
Example	`#INCLUDE moab.acct` *Moab will process the parameters in `moab.acct` and `moab.cfg`.*

INSIGHTENDPOINT

Format	`<hostname>[:<port>]`
Default	---
Description	Enables Moab Workload Manager to connect to Moab Insight. <hostname> is the server where Insight is located. <hostname> is required, <port> is optional.

INSTANTSTAGE

Description	This parameter is deprecated and may be removed in a future release. Use JOBMIGRATEPOLICY.

INVALIDFSTREEMSG	
Format	`"<STRING>"`
Default	`"no valid fstree node found"`
Description	The error message that should be attached to jobs that cannot run because of a fairshare tree configuration violation.
Example	`INVALIDFSTREEMSG "account is invalid for requested partition"`

JOBACTIONONNODEFAILURE	
Format	CANCEL, FAIL, HOLD, IGNORE, NOTIFY, or REQUEUE
Default	---
Description	By default, Moab does not report information when a node allocated to an active job has failed (state is down). Use this parameter to specify the action to take if Moab detects that a node allocated to an active job has failed. Moab only reports this information via diagnostic commands. If this parameter is set, Moab will cancel or requeue the active job. See 4.4.6 Allocated Resource Failure Policy for Jobs for more information. **Note**: The `HOLD` value is only applicable when using checkpointing.
Example	`JOBACTIONONNODEFAILURE REQUEUE` *Moab will requeue active jobs that have allocated nodes that have failed during the execution of the job.*

JOBAGGREGATIONTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`0`
Description	The minimum amount of time the scheduler should wait after receiving a job event until it should process that event. This parameter enables sites with bursty job submissions to process job events in groups decreasing total job scheduling cycles and allowing the scheduler to make more intelligent choices by aggregating job submissions and choosing between the jobs. See Appendix I: Considerations for Large Clusters for more information.

Example	`JOBAGGREGATIONTIME 00:00:04`
	`RMPOLLINTERVAL     30,30`
	Moab will wait `4 seconds` between scheduling cycles when job events have been received and will wait `30 seconds` between scheduling cycles otherwise.

JOBCFG	
Format	`<ATTR>=<VAL>` Where `<ATTR>` is one of ACCOUNT, CLASS, CPUCLOCK, CPULIMIT, DESCRIPTION, DPROCS, ENV, EXEC, FLAGS, GNAME, GRES, GROUP, MEM, NODEACCESSPOLICY, NODES, NODESET, PARTITION, PREF, PRIORITY, PRIORITYF, QOS, RARCH, RFEATURES, RM, ROPSYS, SELECT, SYSTEMJOBTYPE, TASKS, TASKPERNODE, TEMPLATEDEPEND, UNAME, USER, VARIABLE, WCLIMIT
Default	---
Description	Specifies attributes for jobs that satisfy the specified profile. The `SELECT` attribute allows users to specify the job template by using msub -l template=. The `JOBCFG` parameter supports the following attributes: `ACCOUNT`, `CLASS`, `CPUCLOCK`, `CPULIMIT`, `DESCRIPTION`, `DPROCS`, `ENV`, `EXEC`, `FLAGS`, `GNAME`, `GRES`, `GROUP`, `MEM`, `NODEACCESSPOLICY`, `NODES`, `NODESET`, `PARTITION`, `PREF`, `PRIORITY`, `PRIORITYF`, `QOS`, `RARCH`, `RFEATURES`, `RM`, `ROPSYS`, `SELECT`, `SYSTEMJOBTYPE`, `TASKS`, `TASKPERNODE`, `TEMPLATEDEPEND`, `UNAME`, `USER`, `VARIABLE`, `WCLIMIT` It also supports the following Wiki attributes: `ARGS`, `DMEM`, `DDISK`, `DWAP`, `ERROR`, `EXEC`, `EXITCODE`, `GATTR`, `GEVENT`, `IWD`, `JNAME`, `NAME`, `PARTITIONMASK`, `PRIORITYF`, `RDISK`, `RSWAP`, `RAGRES`, `RCGRES`, `TASKPERNODE`, `TRIGGER`, `VARIABLE`, `NULL` **Note**: The *index* to the `JOBCFG` parameter can either be an admin-chosen job template name or the exact name of job reported by one or more workload queries. See N.2.2 Query Workload Data Format and 22.2.1 Job Template Extension Attributes.
Example	`JOBCFG[sql] RFEATURES=sqlnode QOS=service` *When the `sql` job is detected, it will have the specified default qos and node feature attributes set.*

JOBCPURGETIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`00:05:00`
Description	The amount of time Moab will preserve detailed information about a completed job (see 3.7.31 showq and 3.7.1 checkjob).
Example	`JOBCPURGETIME 02:00:00` *Moab will maintain detailed job information for `2 hours` after a job has completed.*

JOBCTRUNCATENLCP	
Format	`<BOOLEAN>`
Default	`TRUE`
Description	Specifies whether Moab will store only the first node of the node list for a completed job in the checkpoint file.
Example	`JOBCTRUNCATENLCP TRUE` *`JOBCTRUNCATENLCP` reduces the amount of memory Moab uses to store completed job information.*

JOBEXTENDSTARTWALLTIME	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Extends the job walltime when Moab starts the job up to the lesser of the maximum or the next reservation (rounded down to the nearest minute). JOBEXTENDSTARTWALLTIME TRUE and JOBEXTENDDURATION cannot be configured together. If they are in the same moab.cfg or are both active, then the JOBEXTENDDURATION will not be honored.

Example	`JOBEXTENDSTARTWALLTIME TRUE`
	Submit job with a minimum wallclock limit and a walltime; for example:
	`echo sleep 500 | msub -A ee -l nodes=5,minwclimit=5:00,walltime=30:00,partition=g02`
	At job start, Moab recognizes the nodes assigned to the specified job and extends the walltime for the job (one time at job start) up to the lesser of the maximum walltime requested or the least amount of time available for any of the nodes until the next reservation on that node.

JOBFAILRETRYCOUNT	
Format	`<INTEGER>`
Default	`0`
Description	The number of times a job is requeued and restarted by Moab if the job fails (if the job itself returns a non-zero exit code). Some types of jobs might succeed if automatically retried several times in short succession. This parameter was created with these types of jobs in mind. Note that the job in question must also be restartable (the job needs to have the 'RESTARTABLE' flag set on it) and the resource manager managing the job must support requeuing and starting completed jobs. If a job fails too many times, and reaches the number of retries given by JobFailRetryCount, then a UserHold is placed on the job and a message is attached to it signifying that the job has a 'restart count violation.'
Example	`JOBFAILRETRYCOUNT 7` *Any job with a RESTARTABLE flag is requeued, if it fails, up to* 7 *times before a UserHold is placed on it.*

JOBIDWEIGHT	
Format	`<INTEGER>`
Default	---
Description	The weight to be applied to the job's ID. See 4.1.2.H Job Attribute (ATTR) Component.
Example	`JOBIDWEIGHT -1` *Later jobs' priority will be negatively affected.*

JOBMATCHCFG	
Format	`<ATTR>=<VAL>` Where `<ATTR>` is one of JMIN, JMAX, JDEF, JSET, or JSTAT
Default	---
Description	The job templates that must be matched and that will be applied in the case of a match.
Example	`JOBMATCHCFG[sql] JMIN=interactive JSTAT=istat`

JOBMAXHOLDTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	0 (meaning, no max hold time)
Description	The amount of time a job can be held before it is canceled automatically.
Example	`JOBMAXHOLDTIME 02:00:00` *Moab will keep jobs in any HOLD state for `2 hours` before canceling them.*

JOBMAXNODECOUNT	
Format	`<INTEGER>`
Default	`1024`
Description	The maximum number of nodes that can be allocated to a job. After changing this parameter, Moab must be restarted. **Note**: This value cannot exceed either **MMAX_NODE** or **MMAX_TASK_PER_JOB**. If larger values are required, these values must also be increased. Moab must be restarted before changes to this command will take effect. The command mdiag -S will indicate if any job node count overflows have occurred. See Appendix I: Considerations for Large Clusters. Moab only reads in this setting when starting up (or restarting).
Example	`JOBMAXNODECOUNT 4000`

JOBMAXOVERRUN

Format	`[[[DD:]HH:]MM:]SS,][[[DD:]HH:]MM:]SS`
Default	(no soft limit), `10 minutes` (hard limit)
Description	Soft and hard limit of the amount of time Moab will allow a job to exceed its wallclock limit before it first sends a mail to the primary admin (soft limit) and then terminates the job (hard limit). See the parameter WCVIOLATIONACTION or 5.2.6 Usage-Based Limits. If you run Moab with the Torque resource manager, you must set the `$ignwalltime` parameter to `true` in the `/var/spool/torque/mom_priv/config` file; otherwise the pbs_mom will kill any job that exceeds its walltime. See '$ignwalltime' in the *Torque Resource Manager Administrator Guide* for more information.
Example	`JOBMAXOVERRUN 15:00,1:00:00` *Jobs may exceed their wallclock limit by up to `1 hour`, but Moab will send an email to the primary administrator when a job exceeds its walltime by `15 minutes`.*

JOBMAXPREEMPTCOUNT

Format	`<INTEGER>`
Default	`0` (No Limit)
Description	Maximum number of times a job can be preempted before it is no longer preemptible.
Example	`JOBMAXPREEMPTCOUNT 5` *Any job may be preempted up to 5 times, after which it is no longer preemptible.*

JOBMAXPREEMPTPERITERATION	
Format	`<INTEGER>`
Default	`0` (No Limit)
Description	Maximum number of jobs allowed to be preempted per iteration.
Example	`JOBMAXPREEMPTPERITERATION 10`

JOBMAXSTARTPERITERATION	
Format	`<INTEGER>`
Default	`0` (No Limit)
Description	Maximum number of jobs allowed to start per iteration.
Example	`JOBMAXSTARTPERITERATION 10`

JOBMAXSTARTTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`-1` (NO LIMIT)
Description	Length of time a job is allowed to remain in a 'starting' state. If a 'started' job does not transition to a running state within this amount of time, Moab will cancel the job, believing a system failure has occurred.
Example	`JOBMAXSTARTTIME 2:00:00` *Jobs may attempt to start for up to* `2 hours` *before being canceled by the scheduler*

JOBMAXTASKCOUNT	
Format	`<INTEGER>`
Default	`32768`
Description	The total number of tasks allowed per job.
Example	`JOBMAXTASKCOUNT 226000`

JOBMIGRATEPOLICY	
Format	One of the following: `IMMEDIATE`, `JUSTINTIME`, or `AUTO`.
Default	`AUTO`
Description	Upon using the msub command to submit a job, you can allocate the job to immediately (`IMMEDIATE`) migrate to the resource manager, or you can instruct Moab to only migrate the job to the resource manager when it is ready to run (`JUSTINTIME`). Specifying `AUTO` allows MOAB to determine on a per-job basis whether to use `IMMEDIATE` or `JUSTINTIME`.
Example	`JOBMIGRATEPOLICY JUSTINTIME`

JOBNAMEWEIGHT	
Format	`<INTEGER>`
Default	---
Description	The weight to be applied to the job's name if the Name contains an integer. See 4.1.2.H Job Attribute (ATTR) Component.
Example	`JOBNAMEWEIGHT 1`

JOBNODEMATCHPOLICY

Format	`AUTO`, `EXACTNODE`, or `EXACTPROC`
Default	`AUTO`
Description	Specifies additional constraints on how compute nodes are to be selected: • `AUTO` overrides the JOBNODEMATCHPOLICY (packs the jobs on any node). • `EXACTNODE` indicates that Moab should select as many nodes as requested even if it could pack multiple tasks onto the same node. • `EXACTPROC` indicates that Moab should select only nodes with exactly the number of processors configured as are requested per node even if nodes with excess processors are available.
Example	`JOBNODEMATCHPOLICY EXACTNODE` *In a PBS/Native job with resource specification `nodes=<x>:ppn=<y>`, Moab will allocate exactly <y> task on each of <x> distinct nodes.*

JOBPREEMPTMAXACTIVETIME

Format	`[[[DD:]HH:]MM:]SS`
Default	`0`
Description	The amount of time when a job can be eligible for preemption. See Chapter 21: Preemption.
Example	`JOBPREEMPTMAXACTIVETIME 00:05:00` *A job is preemptable for the first `5 minutes` of its run time.*

JOBPREEMPTMINACTIVETIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`0`
Description	The minimum amount of time a job must be active before being considered eligible for preemption. See Chapter 21: Preemption.
Example	`JOBPREEMPTMINACTIVETIME 00:05:00` *A job must execute for `5 minutes` before Moab will consider it eligible for preemption.*

JOBPRIOACCRUALPOLICY	
Format	`ACCRUE` or `RESET`
Default	`ACCRUE`
Description	Specifies how Moab should track the dynamic aspects of a job's priority. `ACCRUE` indicates that the job will accrue queuetime based priority from the time it is submitted unless it violates any of the policies not specified in the parameter JOBPRIOEXCEPTIONS. `RESET` indicates that it will accrue priority from the time it is submitted unless it violates any of the JOBPRIOEXCEPTIONS. However, with `RESET`, if the job does violate JOBPRIOEXCEPTIONS then its queuetime based priority will be reset to 0. JOBPRIOACCRUALPOLICY is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]   JOBPRIOACCRUALPOLICY=ACCRUE` The following old `JOBPRIOACCRUALPOLICY` values have been **deprecated** and should be adjusted to the following values: • `QUEUEPOLICY` = `ACCRUE` and `JOBPRIOEXCEPTIONS SOFTPOLICY, HARDPOLICY` • `QUEUEPOLICYRESET` = `RESET` and `JOBPRIOEXCEPTIONS SOFTPOLICY, HARDPOLICY` • `ALWAYS` = `ACCRUE` and `JOBPRIOEXCEPTIONS ALL` • `FULLPOLICY` = `ACCRUE` and `JOBPRIOEXCEPTIONS NONE` • `FULLPOLICYRESET` = `RESET` and `JOBPRIOEXCEPTIONS NONE`
Example	`JOBPRIOACCRUALPOLICY    RESET` *Moab will adjust the job's dynamic priority subcomponents (i.e., QUEUETIME, XFACTOR, and TARGETQUEUETIME, etc. each iteration that the job does not violate any `JOBPRIOEXCEPTIONS`, if it is found in violation, its queuetime will be reset to 0).*

JOBPRIOEXCEPTIONS	
Format	Comma-delimited list of any of the following: DEFER, DEPENDS, SOFTPOLICY, HARDPOLICY, IDLEPOLICY, USERHOLD, BATCHHOLD, and SYSTEMHOLD (ALL or NONE can also be specified on their own).
Default	NONE
Description	Specifies exceptions for calculating a job's dynamic priority (QUEUETIME, XFACTOR, TARGETQUEUETIME). Normally, when a job violates a policy, is placed on hold, or has an unsatisfied dependency, it will not accrue priority. Exceptions can be configured to allow a job to accrue priority in spite of any of these violations. With DEPENDS a job will increase in priority even if there exists an unsatisfied dependency. With SOFTPOLICY, HARDPOLICY, or IDLEPOLICY a job can accrue priority despite violating a specific limit. With DEFER, USERHOLD, BATCHHOLD, or SYSTEMHOLD a job can accrue priority despite being on hold. JOBPRIOEXCEPTIONS is a global parameter, but can be configured to work only in QOSCFG: `QOSCFG[arrays]  JOBPRIOEXCEPTIONS=IDLEPOLICY`
Example	`JOBPRIOEXCEPTIONS BATCHHOLD,SYSTEMHOLD,DEPENDS` *Jobs will accrue priority in spite of batchholds, systemholds, or unsatisfied dependencies.*

JOBPRIOF	
Format	<ATTRIBUTE>[<VALUE>]=<PRIORITY> Where <ATTRIBUTE> is one of ATTR, GRES, or STATE.
Default	---
Description	Specifies attribute priority weights for jobs with specific attributes, generic resource requests, or states. State values must be one of the standard Moab job states. See 4.1.2.H Job Attribute (ATTR) Component.

Example	`JOBPRIOF        STATE[Running]=100  STATE[Suspended]=1000  ATTR[PREEMPTEE]=200 GRES[biocalc]=5`
	`ATTRATTRWEIGHT    1`
	`ATTRSTATEWEIGHT   1`
	Moab will adjust the job's dynamic priority subcomponents.

JOBPURGETIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`30`
Description	The amount of time Moab will keep a job record that is no longer reported by the resource manager. Useful when using a resource manager that*drops* information about a job due to internal failures. See the parameter JOBCPURGETIME. Set to 0 to purge immediately if the resource manager does not report the job.
Example	`JOBPURGETIME 00:05:00` *Moab will maintain a job record for `5 minutes` after the last update regarding that object received from the resource manager.*

JOBREJECTPOLICY	
Format	One or more of `CANCEL`, `HOLD`, `IGNORE`, `MAIL`, or `RETRY`.
Default	`HOLD`
Description	The action to take when the scheduler determines that a job can never run. `CANCEL` issues a call to the resource manager to cancel the job. `HOLD` places a *batch* hold on the job preventing the job from being further evaluated until released by an admin. **Note**: Admins can dynamically alter job attributes and possibly *fix* the job with mjobctl -m. With `IGNORE`, the scheduler will allow the job to exist within the resource manager queue but will neither process it nor report it. MAIL will send email to both the admin and the user when rejected jobs are detected. If `RETRY` is set, then Moab will allow the job to remain idle and will only attempt to start the job when the policy violation is resolved. Any combination of attributes can be specified. See the parameter QOSREJECTPOLICY.
Example	`JOBREJECTPOLICY  MAIL,CANCEL`

JOBREMOVEENVVARLIST	
Format	Comma-delimited list of strings.
Default	---
Description	Moab will remove the specified environment variables from the job's environment before migrating the job to its destination resource manager. This is useful when jobs submit themselves from one cluster to another with the full environment. This parameter is currently only supported with Torque resource managers.
Example	`JOBREMOVEENVVARLIST PBS_SERVER,TZ` *Moab will remove the environment variables `PBS_SERVER` and `TZ` before submitting jobs.*

JOBRETRYTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`00:00:60`
Description	Period of time Moab will continue to attempt to start a job that has failed to start due to transient failures or that has successfully started and was then rejected by the resource manager due to transient failures. For related information, see 6.1.4 Reservation Policies, and the parameters DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, NODEFAILURERESERVETIME, and GUARANTEEDPREEMPTION.
Example	`JOBRETRYTIME    00:05:00` *Moab will try for up to `5 minutes` to restart jobs if the job start has failed due to transient errors.*

LIMITEDJOBCP	
Format	`<BOOLEAN>`
Default	`TRUE`
Description	By default, Moab does not update the checkpoint file or the cache for a job unless it has undergone a qualifying change (such as a change of state). Setting `LIMITEDJOBCP` to `FALSE` will cause Moab to update the checkpoint file and the cache for all jobs each iteration, even without a qualifying change. For clusters that routinely run large numbers of jobs (e.g., more than 15,000), setting this parameter to `FALSE` might adversely affect iteration times.
Example	`LIMITEDJOBCP FALSE` *Moab will update the checkpoint file and cache for all jobs each iteration.*

LIMITEDNODECP	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether there should be limited node checkpointing (see Appendix I: Considerations for Large Clusters
Example	`LIMITEDNODECP TRUE` *Moab will only maintain scheduler checkpoint information for nodes with explicitly modified job attributes (some minor node performance and usage statistics may be lost).*

LOADALLJOBCP	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether Moab should load, during startup, all non-completed jobs in the checkpoint files regardless of whether or not their corresponding resource managers are active. For example, this allows source peers to continue showing remote jobs in the queue based on checkpointed information, even though the destination peer is offline.
Example	`LOADALLJOBCP TRUE` *Moab will load, at startup, all non-completed jobs from all checkpoint files.*

LOCKFILE	
Format	`<STRING>`
Default	---
Description	The path for the lock (pid) file used by Moab.
Example	`LOCKFILE /var/spool/moab/lock`

LOGDIR	
Format	`<STRING>`
Default	log
Description	The directory where log files will be maintained. If specified as a relative path, `LOGDIR` will be relative to `$(MOABHOMEDIR)`. See 12.2 Logging for more information.
Example	`LOGDIR /var/spool/moab` *Moab will record its log files directly into the `/var/spool/moab` directory.*

LOGFACILITY	
Format	Colon-delimited list of one or more of the following: `CORE`, `SCHED`, `SOCK`, `UI`, `LL`, `CONFIG`, `STAT`, `SIM`, `STRUCT`, `FS`, `CKPT`, `BANK`, `RM`, `PBS`, `WIKI`, `ALL`.
Default	`ALL`
Description	Specifies which types of events to log (see 12.2 Logging).
Example	`LOGFACILITY RM:PBS` *Moab will log only events involving general resource manager or PBS interface activities.*

LOGFILE	
Format	`<STRING>`
Default	`moab.log`
Description	Name of the Moab log file. This file is maintained in the directory pointed to by `<LOGDIR>` unless `<LOGFILE>` is an absolute path (see 12.2 Logging).
Example	`LOGFILE moab.test.log` *Log information will be written to the file `moab.test.log` located in the directory pointed to by the LOGDIR parameter.*

LOGFILEMAXSIZE	
Format	`<INTEGER>`
Default	`10000000`
Description	Maximum allowed size (in bytes) of the log file before it will be rolled (see 12.2 Logging).
Example	`LOGFILEMAXSIZE 50000000` *Log files will be rolled when they reach `50 MB` in size*

LOGFILEROLLDEPTH

Format	`<INTEGER>`
Default	`3`
Description	Number of old log files to maintain (i.e., when full, `moab.log` will be renamed `moab.log.1`, `moab.log.1` will be renamed `moab.log.2`, ...). See 12.2 Logging.
Example	`LOGFILEROLLDEPTH 5` *Moab will maintain and roll the last* `5` *log files.*

LOGLEVEL

Format	`<INTEGER>` (0-9)
Default	`0`
Description	The verbosity of Moab logging where 9 is the most verbose (**Note**: each logging level is approximately an order of magnitude more verbose than the previous level). See 12.2 Logging.
Example	`LOGLEVEL 4` *Moab will write all Moab log messages with a threshold of* `4` *or lower to the* `moab.log` *file.*

LOGLEVELOVERRIDE	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	When this parameter is on, if someone runs a command with `--loglevel=<x>`, that loglevel, if higher than the current loglevel, is used on the scheduler side for the duration of the command. All logs produced during that time are put into a separate log file (this creates a 'gap' in the normal logs). This can be very useful for debugging, but we recommend that this be used only when diagnosing a specific problem so that users can't affect performance by submitting multiple `--loglevel` commands. This parameter does not work with threaded commands (such as *`showq`*, *`mdiag -n`*, and *`mdiag -j`*).
Example	`LOGLEVELOVERRIDE    TRUE`

LOGPERMISSIONS	
Format	`<INTEGER>`
Default	`644`
Description	The octal number that represents read, write, and execute permissions.
Example	`LOGPERMISSIONS 600` *Allows the file owner to read and write permissions, but denies rights to the group and others.*

LOGROLLACTION	
Format	`<STRING>`
Default	---
Description	Specifies a script to run when the logs roll. The script is run as a trigger and can be viewed using mdiag -T. For example, a script can be specified that always moves the first rolled log file, `moab.log.1`, to an archive directory for longer term storage.
Example	`LOGROLLACTION /usr/local/tools/logroll.pl`

MAILFROMADDR	
Format	`<EMAILADDRESS>`
Default	---
Description	Sets the FROM address for all emails sent from Moab. Used in conjunction with MAILPROGRAM.
Example	`MAILFROMADDR it@yourdomain.com`

MAILPROGRAM	
Format	`[<Full_Path_To_Mail_Command> | DEFAULT | NONE ][@<DEFAULTMAILDOMAIN>]`
Default	`NONE`
Description	If set to `NONE`, no mail is sent. If set to `DEFAULT`, Moab sends mail via the system's default mail program (usually `/usr/bin/sendmail`). If set to the local path of a mail program, Moab uses the specified mail program to send mail. By default, Moab mail notification is disabled. To enable, you must set `MAILPROGRAM` to `DEFAULT` or specify some other locally available mail program. If the *default mail domain* is set, emails will be routed to this domain unless a per-user domain is specified using the `EMAILADDRESS` attribute of the USERCFG parameter. If neither of these values is set, Moab uses '@localhost' as the mail domain. See 12.4 Notifying Administrators of Failures. For jobs, the email address used on the msub -M option overrides all other user email addresses. Additionally, admins are notified in the case of job violations.

Example	`MAILPROGRAM DEFAULT`
	
	Moab sends mail via the system's default mail program, `/usr/bin/sendmail`.
	
	`MAILPROGRAM /usr/local/bin/sendmail@mydomain.com`
	
	Moab sends mail via the mail program located at `/usr/local/bin/sendmail` with default mail domain `@mydomain.com`

MAXGRES	
Format	`<INTEGER>`
Default	`512`
Description	Specifies how many generic resources Moab should manage. Moab only reads in this setting when starting up (or restarting).
Example	`MAXGRES 1024`

MAXGMETRIC	
Format	<INTEGER>
Default	10
Description	Specifies how many generic metrics Moab should manage. Moab only reads in this setting when starting up (or restarting).
Example	`MAXGMETRIC 20`

MAXGREENSTANDBYPOOLSIZE	
Format	<INTEGER>
Default	-1
Description	Set the size of the standby pool. This is the number of idle nodes that will be powered on and idle. As the workload changes, Moab turns nodes on or off to try to meet this goal. A value of -1 disables the standby pool. See 15.4 Enabling Green Computing.
Example	`MAXGREENSTANDBYPOOLSIZE 5` *Moab keeps up to 5 idle nodes powered on to be kept on standby.*

MAXJOB	
Format	`<INTEGER>`
Default	`51200`
Description	The maximum quantity of jobs for which Moab should allocate memory used for tracking jobs. If Moab is tracking the maximum quantity of jobs specified by this parameter, it rejects subsequent jobs submitted by any user since it has no memory left with which to track newly submitted jobs. If a user submitted a job with the *`msub`* command, this rejection behavior requires the user to resubmit the job at a later time after other jobs have completed, which frees memory in which Moab can place later-submitted jobs. If a user submitted a job with the Torque qsub command, Torque will automatically resubmit the job to Moab until Moab accepts it. The mdiag -S command indicates if any job overflows have occurred. If this parameter's value is changed, it does not go into effect until Moab restarts. Moab reads the parameter only on initial startup and uses its value to allocate the memory it uses to track jobs. Moab only reads in this setting when starting up (or restarting).
Example	`MAXJOB 75000`

MAXNODE	
Format	`<INTEGER>`
Default	`5120`
Description	The maximum number of compute nodes supported. Moab only reads in this setting when starting up (or restarting).
Example	`MAXNODE 10000`

MAXRSVPERNODE	
Format	`<INTEGER>`
Default	`64`
Description	The maximum number of reservations on a node. For large SMP systems (>512 processors/node), Adaptive Computing advises adjusting the value to approximately twice the average sum of admin, standing, and job reservations present. A second number, led by a comma, can also be specified to set a maximum number of reservations for nodes that are part of the SHARED partition. The maximum possible value of `MAXRSVPERNODE` is `8192` for a global node and `4096` for any other node. Moab must be restarted for any changes to this parameter to take effect. The command mdiag -S indicates whether any node reservation overflows have occurred. See Appendix I: Considerations for Large Clusters. Do not lower the MAXRSVPERNODE value while there are active jobs in the queue. This can lead to queue instability and certain jobs could become stuck or disconnected from the system. Moab only reads in this setting when starting up (or restarting).

Example	`MAXRSVPERNODE 64`
	`64` is the maximum number of reservations on a single node.
	`MAXRSVPERNODE 100,7000`
	`100` is the maximum number of reservations on a single node, and `7000` is the maximum number of reservations for global nodes.

MEMREFRESHINTERVAL	
Format	`[[[DD:]HH:]MM:]:SS` | job:`<COUNT>`
Default	---
Description	The time interval or total job query count at which Moab will perform garbage collection to free memory associated with resource manager APIs that possess memory leaks (i.e., Loadleveler, etc.).

Example	`# free memory associated with leaky RM API`
	`MEMREFRESHINTERVAL 24:00:00`
	Moab will perform garbage collection once every `24 hours`.

MEMWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The coefficient to be multiplied by a job's MEM (dedicated memory in MB) factor. See 4.1.2.D Resource (RES) Component.

Example	`RESWEIGHT 10`
	`MEMWEIGHT 1000`
	*Each job's priority will be increased by `10` * `1000` * `<request memory>`.*

MESSAGEQUEUEADDRESS	
Format	The IP address of the machine on which Moab is generating events.
Default	`*` (all)
Description	When a user subscribes to the events Moab provides and delivers via zeroMQ, the user must do so by specifying `tcp://<ipAddress>:<port>`. `MESSAGEQUEUEADDRESS` specifies the `<ipAddress>`, which must match the IP address of the machine on which Moab is installed. To specify the port, see the parameter MESSAGEQUEUEPORT.
Example	`MESSAGEQUEUEADDRESS    10.1.0.10` *To subscribe to Moab events, users must use `tcp://10.1.0.10:<port>`.*

MESSAGEQUEUEPORT	
Format	The port of the machine on which Moab is generating events.
Default	`5563`
Description	When a user subscribes to the events Moab provides and delivers via zeroMQ, the user must do so by specifying `tcp://<ipAddress>:<port>`. `MESSAGEQUEUEPORT` specifies the `<port>`, which must match the port of the machine on which Moab is installed. To specify the IP address, see the parameter MESSAGEQUEUEADDRESS.
Example	`MESSAGEQUEUEPORT    1010` *To subscribe to Moab events, users must use `tcp://<ipAddress>:1010`.*

MESSAGEQUEUESECRETKEY	
Format	`<STRING>`
Default	---
Description	Causes Moab to encrypt the events delivered via zeroMQ using the Advanced Encryption Standard (AES) algorithm. Must be a Base64-encoded, 128-bit (16-byte) key. Messages will be encrypted using AES in CBC mode where inputs are padded with PKCS5 padding. The initialization vector is calculated by using an MD5 hash of the key specified in `MESSAGEQUEUESECRETKEY`. `MESSAGEQUEUESECRETKEY` can only be specified in the `moab-private.cfg` file.
Example	`MESSAGEQUEUESECRETKEY 1r6RvfqJa6voezy5wAx0hw==`

MINADMINSTIME	
Format	`<INTEGER>`
Default	`60` seconds
Description	The minimum time a job will be suspended if suspended by an admin or by a scheduler policy.
Example	`MINADMINSTIME 00:10:00` *Each job suspended by admins or policies will stay in the suspended state for at least `10 minutes`.*

MINPRIORITYJOBRSVSIZE	
Format	`<INTEGER>`
Default	`0`
Description	The minimum total job size, in processors, for a job to receive a priority reservation. Jobs smaller than this value will still be started during normal and backfill scheduling, but will not be eligible for priority reservations.
Example	`MINPRIORITYJOBRSVSIZE 4` *Any job requesting less than four processors will not receive a priority reservation.*

MISSINGDEPENDENCYACTION	
Format	`CANCEL`, `HOLD`, or `RUN`
Default	`HOLD`
Description	Controls what Moab does with a dependent job when its dependency job cannot be found when Moab evaluates the dependent job for scheduling. This only affects jobs whose dependent job cannot be found.
Example	`MISSINGDEPENDENCYACTION CANCEL` *Any job that has a dependent job that cannot be found is canceled.*

MONGOREPLICASETNAME	
Format	`<name>`
Default	---
Description	If MONGOSERVER is a comma-separated list of HostAndPort strings (i.e., replicaset), MONGOREPLICASETNAME must be set to the name used when defining the replica set within MongoDB.
Example	see below

```
rs.initiate(
   {
      _id: "myReplSet",
      version: 1,
      members: [
         { _id: 0, host : "mongodb0.example.net:27017" },
         { _id: 1, host : "mongodb1.example.net:27017" },
         { _id: 2, host : "mongodb2.example.net:27017" }
      ]
   }
)
```

```
MONGOSERVER
mongodb0.example.net:27017,mongodb1.example.net27017,mongodb2.example.net:27017
MONGOREPLICASETNAME myReplSet
MONGOSSLMODE enabled
MONGOSSLCAFILE /etc/ssl/mongodb-cert.crt
```

MONGOSERVER	
Format	`<server>[:<port>]`
Default	---
Description	The MongoDB server DNS or IP and optional port number. The port number defaults to the MongoDB default (27017) when not given.
Example	`MONGOSERVER localhost:27017`

MONGOSSLCAFILE	
Format	`<file>`
Default	---
Description	`<file>` is a file containing the public cert located in the net.ssl.PEMKeyFile defined in the mongod.conf (i.e., /etc/mongod.conf) file on the MongoDB host. MONGOSSLCAFILE is ignored when MONGOSSLMODE is either undefined or set to disabled.

Example	`MONGOSERVER  localhost:27017`
	`MONGOSSLMODE  enabled`
	`MONGOSSLCAFILE  /etc/ssl/mongodb-cert.crt`

MONGOSSLMODE	
Format	`enabled` or `disabled`
Default	`disabled`
Description	Enables or disables encryption of MongoDB network traffic. MONGOSSLCAFILE is required when MONGOSSLMODE is set to 'enabled'. 'preferred' can be added if the Mongo driver supports it as an option.

Example	`MONGOSERVER localhost:27017`
	`MONGOSSLMODE enabled`
	`MONGOSSLCAFILE /etc/ssl/mongodb-cert.crt`

MSUBQUERYINTERVAL	
Format	`<INTEGER>`
Default	`5` seconds
Description	The length of the interval (in seconds) between job queries when using msub -K. Jobs submitted with the -K option query the scheduler every `MSUBQUERYINTERVAL` seconds until the job is completed. `MSUBQUERYINTERVAL` can exist as an environment variable. Any value in `moab.cfg` overrides the environment variable. **Note**: If `MSUBQUERYINTERVAL` is set to 0, the -K option will be disabled. Jobs will still submit correctly, but the client will not continue to check on the job.
Example	`MSUBQUERYINTERVAL   60` *If a user uses the msub -K command, the client remains open and queries the server every `60 seconds` until the job completes.*

NODEACCESSPOLICY	
Format	One of the following: SHARED, SHAREDONLY, SINGLEACCOUNT, SINGLECLASS, SINGLEGROUP, SINGLEJOB, SINGLETASK, SINGLEUSER, or UNIQUEUSER
Default	**SHARED**
Description	Specifies how node resources will be shared by various tasks (see 4.3 Node Access Policies for more information).
Example	`NODEACCESSPOLICY SINGLEUSER` Moab will allow resources on a node to be used by more than one job provided that the jobs are all owned by the same user.

NODEAFFINITYPOLICY	
Format	POSITIVE or DEFAULT
Default	**DEFAULT**
Description	When multiple reservations are on the same node and a job has access to some with a positive affinity *and* to others with a negative affinity, then the last reservation's affinity wins (by default). When NODEAFFINITYPOLICY is set to POSITIVE *and* a job has any positive affinity on the node, then the positive affinity will have precedent over any negative affinity.
Example	`NODEAFFINITYPOLICY POSITIVE` If a job has any positive affinity to a node, it will take precedent over any negative affinity.

NODEALLOCATIONPOLICY	
Format	One of the following: CONTIGUOUS, CPULOAD, FIRSTAVAILABLE, LASTAVAILABLE, MAXBALANCE, MINRESOURCE, PRIORITY, or PLUGIN
Default	LASTAVAILABLE
Description	Specifies how Moab should allocate available resources to jobs. See 4.2 Node Allocation Policies for more information. If ENABLEHIGHTHROUGHPUT is TRUE, you must set NODEALLOCATIONPOLICY to FIRSTAVAILABLE.
Example	`NODEALLOCATIONPOLICY MINRESOURCE` *Moab will apply the node allocation policy* MINRESOURCE *to all jobs by default.*

NODEALLOCRESFAILUREPOLICY	
Format	One of the following: CANCEL, HOLD, IGNORE, MIGRATE, NOTIFY, or REQUEUE.
Default	NONE
Description	Specifies how Moab should handle active jobs that experience node failures during execution. See the RESFAILPOLICY resource manager extension or 4.4 Node Availability Policies.
Example	`NODEALLOCRESFAILUREPOLICY REQUEUE` *Moab will requeue jobs that have allocated nodes fail during execution.*

NODEAVAILABILITYPOLICY	
Format	`<POLICY>[:<RESOURCETYPE>]` ... Where `<POLICY>` is one of `COMBINED`, `DEDICATED`, or `UTILIZED` and `<RESOURCETYPE>` is one of `PROC`, `MEM`, `SWAP`, or `DISK`.
Default	`COMBINED`
Description	Specifies how available node resources are reported. Moab uses the following calculations to determine the amount of available resources: • `Dedicated` (use what Moab has scheduled to be used): Available = Configured - Dedicated • `Utilized` (use what the resource manager is reporting is being used): Available = Configured - Utilized • `Combined` (use the larger of dedicated and utilized): Available = Configured - (MAX (Dedicated, Utilized)) Moab marks a node as busy when it has no available processors, so `NODEAVAILABILTYPOLICY`, by affecting how many processors are reported as available, also affects node state. See 4.4 Node Availability Policies for more information. You can also set NODEAVAILABILITYPOLICY at NODECFG. See the attribute NODEAVAILABILITYPOLICY for instructions on setting this at the local level.
Example	`NODEAVAILABILITYPOLICY DEDICATED:PROC COMBINED:MEM` *Moab will ignore resource utilization information in locating available processors for jobs but will use both dedicated and utilized memory information in determining memory availability.*

NODEBUSYSTATEDELAYTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`0:01:00` (one minute)
Description	Length of time Moab will assume busy nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node.
Example	`NODEBUSYSTATEDELAYTIME 0:30:00` *Moab will assume busy nodes are not available for scheduling for at least `30 minutes` from the current time. Therefore, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than `30 minutes` in the future.*

NODECATCREDLIST	
Format	`<LABEL>=<NODECAT>[,<NODECAT>]...[ <LABEL>=<NODECAT>[,<NODECAT>]...]...` Where `<LABEL>` is any string and `<NODECAT>` is one of the defined node categories.
Default	---
Description	If specified, Moab will generate node category groupings and each iteration will assign usage of matching resources to pseudo-credentials with a name matching the specified label. See the Node Categorization section for more information.
Example	`NODECATCREDLIST down=BatchFailure,HardwareFailure,NetworkFailure idle=Idle` *Moab will create a `down` user, group, account, class, and QoS and will associate `BatchFailure`, `HardwareFailure`, and `NetworkFailure` resources with these credentials. Additionally, Moab will assign all `Idle` resources to matching `idle` credentials.*

NODECFG[X]

Format	List of space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: ACCESS, ARCH, CHARGERATE, COMMENT, ENABLEPROFILING, FEATURESGRES, MAXJOB, MAXJOBPERUSER, MAXLOAD, MAXPE, MAXPEPERJOB, MAXPROC, NODEAVAILABILITYPOLICY, NODEINDEX, NODETYPE, OS, OSLIST, PARTITION, POWERPOLICY, PREEMPTMAXCPULOAD, PREEMPTMINMEMAVAIL, PREEMPTPOLICY, PRIORITY, PRIORITYF, PROCSPEED, PROVRM, RACK, RADISK, RCDISK, RCMEM, RCPROC, RCSWAP, SIZE, SLOT, SPEED, TRIGGER, VARIABLE
Default	---
Description	Specifies node-specific attributes for the node indicated in the array field. See Chapter 10: General Node Administration for more information.
Example	`NODECFG[nodeA] MAXJOB=2 SPEED=1.2` *Moab will only allow `2` simultaneous jobs to run on node `nodeA` and will assign a relative machine speed of `1.2` to this node.*

NODEDOWNSTATEDELAYTIME

Format	`[[[DD:]HH:]MM:]SS`
Default	`-1` (never)
Description	Length of time Moab will assume down, drained (offline), or corrupt nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node. The default specification of '-1' causes Moab to never create job reservations on down nodes. See 4.4 Node Availability Policies for more information.
Example	`NODEDOWNSTATEDELAYTIME 0:30:00` *Moab will assume down, drained, and corrupt nodes are not available for scheduling for at least `30 minutes` from the current time. Therefore, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than `30 minutes` in the future.*

NODEDOWNTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	---
Description	The maximum time a previously reported node remains unreported by a resource manager before the node is considered to be in the down state. This can happen if communication with a resource manager or a peer server is lost for more than the specified length of time, or if there is communication with the resource manager but it fails to report the node status.
Example	`NODEDOWNTIME 10:00` *If Moab loses communication with the resource manager for more than `10 minutes`, it sets the state of all nodes belonging to that resource manager to DOWN.*

NODEDRAINSTATEDELAYTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`3:00:00` (three hours)
Description	Length of time Moab will assume drained nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node. Specifying '-1' will cause Moab to never create job reservations on drained nodes. See 4.4 Node Availability Policies for more information.
Example	`NODEDRAINSTATEDELAYTIME 0:30:00` *Moab will assume down, drained, and corrupt nodes are not available for scheduling for at least `30 minutes` from the current time. Therefore, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than `30 minutes` in the future.*

NODEFAILURERESERVETIME

Format	`[[[DD:]HH:]MM:]SS`
Default	`0:05:00`
Description	Duration of reservation Moab will place on any node where it detects a failure from the resource manager (0 indicates no reservation will be placed on the node). See 4.4 Node Availability Policies for more information. See also the attribute NODEFAILURERSVPROFILE. For related information, see 6.1.4 Reservation Policies, and the parameters DEFERSTARTCOUNT, DEFERTIME, RESERVATIONRETRYTIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.
Example	`NODEFAILURERESERVETIME 10:00` *Moab will reserve failed nodes for `10 minutes`.*

NODEIDFORMAT

Format	`<STRING>`
Default	`*$N*`
Description	Specifies how a node ID can be processed to extract possible node, rack, slot, and cluster index information. The value of the parameter can include the markers `$C` (cluster index), `$N` (node index), `$R` (rack index), or `$S` (slot index) separated by `*`(asterisk - representing any number of non-numeric characters) or other characters to indicate this encoding. See 10.2.4 Node Selection for more information on use of node, rack, and slot indices.
Example	`NODEIDFORMAT *$R*$S` *Moab will extract rack and slot information from the cluster node IDs (i.e., `tg-13s08`).*

NODEIDLEPOWERTHRESHOLD

Format	`<INTEGER>`
Default	`60 seconds`
Description	Specifies how long to allow a node to be idle before performing a power action. Increasing the idle duration prevents power on/off thrashing.
Example	`NODEIDLEPOWERTHRESHOLD 300` *Moab will wait 5 minutes before performing a power action on a node that has become idle.*

NODEIDLEPURGETIME

Format	`<SECONDS>`
Default	`0`
Description	When dynamic nodes are created in Moab, they are usually created with a request ID. Dynamic nodes created with a request ID are eligible to be scheduled for purging using NODEIDLEPURGETIME. NODEIDLEPURGETIME is the amount of time for all nodes with the same request ID to be idle before Moab begins firing the node end trigger for each iteration. If one or more of the nodes with the same request ID becomes non-idle, Moab stops firing the node end trigger for all of the nodes with the same request ID until the NODEIDLEPURGETIME is once again met. A value of 0 disables this feature.
Example	`NODEIDLEPURGETIME 300` *Moab will begin purging groups of dynamic nodes with the same request ID when all nodes with the same request ID have been idle for 300 seconds.* *Here is an example of how to create a dynamic node with a request ID:* *qmgr -c "create node elastic_node01 np=16,TTL=2024-6-16T17:17:8Z,requestid=unique_identifierXYZ"*

NODEMAXLOAD	
Format	`<DOUBLE>`
Default	`0.0`
Description	The maximum CPU load on an idle or running node. If the node's load reaches or exceeds this value, Moab will mark the node busy. You can also set the MAXLOAD at NODECFG. However, setting NODECFG MAXLOAD to -1 unsets this parameter setting. See 10.4.1.D MAXLOAD for instructions on setting this at the local level.
Example	`NODEMAXLOAD 0.75` *Moab will adjust the state of all idle and running nodes with a load >= `.75` to the state busy.*

NODEMEMOVERCOMMITFACTOR	
Format	`<DOUBLE>`
Default	---
Description	The parameter overcommits available and configured memory and swap on a node by the specified factor (for example: mem/swap * factor). Used to show that the node has more mem and swap than it really does. Only works for PBS resource manager types.
Example	`NODEMEMOVERCOMMITFACTOR .5` *Moab will overcommit the memory and swap of the node by a factor of `0.5`.*

NODESETATTRIBUTE

Format	FEATURE or VARATTR
Default	---
Description	The type of node attribute by which node set boundaries will be established. See the section Node Set Attribute.

Example	`NODESETPOLICY     ONEOF`
	`NODESETATTRIBUTE  FEATURE`
	Moab will create node sets containing nodes with common features.

NODESETDELAY

Format	[[[DD:]HH:]MM:]SS
Default	0:00:00
Description	Causes Moab to attempt to span a job evenly across nodesets unless doing so delays the job beyond the requested NODESETDELAY. Must use with NODESETPLUS set to SPANEVENLY; if you do not want to use SPANEVENLY, use NODESETISOPTIONAL instead of NODESETDELAY.

Example	`NODESETPLUS SPANEVENLY`
	`NODESETDELAY     5:00`
	Moab tries to span the job evenly across nodesets unless doing so delays the job by 5 minutes.

NODESETISOPTIONAL

Format	`<BOOLEAN>`
Default	`TRUE`
Description	Specifies whether or not Moab will start a job if a requested node set cannot be satisfied. See the section Node Set Constraint Handling.
Example	`NODESETISOPTIONAL TRUE` *Moab will not block a job from running if its node set cannot be satisfied.*

NODESETLIST

Format	`<ATTR>[{ :,|}<ATTR>]...`
Default	---
Description	The list of node attribute values that will be considered for establishing node sets. See the section Node Set List.

Example	`NODESETPOLICY      ONEOF`
	`NODESETATTRIBUTE   FEATURE`
	`NODESETLIST        switchA,switchB`
	Moab will allocate nodes to jobs either using only nodes with the `switchA` *feature or using only nodes with the* `switchB` *feature.*

NODESETPLUS	
Format	`DELAY` or `SPANEVENLY`
Default	---
Description	Specifies how Moab distributes jobs among nodesets. See 7.3.3 NODESETPLUS. Neither `SPANEVENLY` nor `DELAY` values of the NODESETPLUS parameter will work with multi-req jobs or preemption.

Example	`NODESETPLUS SPANEVENLY`
	Moab attempts to fit all jobs on a single nodeset or to span them evenly across a number of nodesets, unless doing so would delay a job beyond the requested NODESETDELAY.
	`NODESETPLUS DELAY`
	Moab attempts to schedule the job within a nodeset for the configured NODESETDELAY. If Moab cannot find space for the job to start within NODESETDELAY (Moab considers future workload to determine if space will open up in time and might create a future reservation), then Moab schedules the job and ignores the nodeset requirement.

NODESETPOLICY	
Format	`ANYOF`, `FIRSTOF`, or `ONEOF`
Default	---
Description	Specifies how nodes will be allocated to the job from the various node set generated. See the section Node Set Policy.

Example	`NODESETPOLICY     ONEOF`
	`NODESETATTRIBUTE  NETWORK`
	Moab will create node sets containing nodes with common network interfaces.

NODESETPRIORITYTYPE

Format	One of `AFFINITY`, `BESTFIT`, `FIRSTFIT`, `WORSTFIT`, or `MINLOSS`.
Default	`FIRSTFIT`
Description	Specifies how resource sets will be selected when more than one feasible resource can be found. See the section Node Set Priority.

Example	`NODESETPRIORITYTYPE BESTFIT`
	`NODESETATTRIBUTE    PROCSPEED`
	Moab will select the resource set that most closely matches the set of resources requested.

NODETOJOBATTRMAP

Format	Comma-delimited list of node features.
Default	---
Description	Job requesting the listed node features will be assigned a corresponding job attribute. These job attributes can be used to enable reservation access, adjust job priority or enable other capabilities. Feature/attribute matching is case-sensitive. In particular, keep in mind that PREEMPTEE and INTERACTIVE require uppercase.
Example	`NODETOJOBATTRMAP  fast,big` *Jobs requesting node feature `fast` or `big` (for instance, with **-l feature=fast** or **-W x=feature:big**) will be assigned a corresponding job attribute.*

NODEUNTRACKEDRESDELAYTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`0:00:00`
Description	Length of time Moab will assume untracked generic resources will remain unavailable for scheduling if a system reservation is not explicitly created for the node. If `NODEUNTRACKEDRESDELAYTIME` is enabled and there is an untracked resource preventing a job from running, then the job remains in the idle queue instead of being deferred.
Example	`NODEUNTRACKEDRESDELAYTIME 0:30:00` *Moab will assume untracked generic resources are not available for scheduling for at least* `30 minutes` *from the current time. Therefore, these nodes will never be allocated to starting jobs. Also, these nodes will only be available for reservations starting more than* `30 minutes` *in the future.*

NODEWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight that will be applied to a job's requested node count before this value is added to the job's cumulative priority. **Notes**: This weight currently only applies when a nodecount is specified by the user job; if the job only specifies tasks or processors, no node factor will be applied to the job's total priority. In order for NODEWEIGHT to function properly, JOBNODEMATCHPOLICY should be set to EXACTNODE. These will be rectified in future versions.
Example	`NODEWEIGHT 1000`

NOLOCALUSERENV

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If `TRUE`, specifies that a user's UserID, GroupID, and HomeDirectory are available on the Moab server host.
Example	`NOLOCALUSERENV TRUE`

NOJOBHOLDNORESOURCES

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If `TRUE`, Moab does not place a hold on jobs that don't have feasible resources. For example, suppose there are 20 processors available for ClassA and 50 processors for the entire system. If a job requests 21 or more processors from ClassA, or 51 or more processors from the entire system, Moab idles the job (instead of putting a hold on it) until the resources become available.
Example	`NOJOBHOLDNORESOURCES TRUE`

NOTIFICATIONPROGRAM

Format	`<STRING>`
Default	---
Description	The name of the program to handle all notification call-outs.
Example	`NOTIFICATIONPROGRAM   tools/notifyme.pl`

NOWAITPREEMPTION

Format	`<BOOLEAN>`
Default	---
Description	Generally when a job is trying to preempt another, it just waits for the original jobs that it chose to preempt to end. If this parameter is on, the preemptor will continue trying to preempt jobs until it can get in.
Example	`NOWAITPREEMPTION   TRUE`

OSCREDLOOKUP

Format	`BESTEFFORT` or `NEVER`
Default	`BESTEFFORT`
Description	When set to `BESTEFFORT`, Moab will create a 'moab user' and try to associate it what a 'system user' (if possible). When set to `NEVER`, this disables all Moab OS credential lookups, including UID, GID, user to group mappings, and any other OS specific information. Setting `OSCREDLOOKUP` by itself does not allow job submission; additional configuration is required. When submitting jobs from user accounts that do not exist on the head node (where Moab and Torque are running), you must also set the PROXYJOBSUBMISSION flag in addition to specifying configuration settings in the resource manager configuration file. See the example that follows for information on required resource manager settings.

Example	`OSCREDLOOKUP NEVER`
	`RMCFG[] FLAGS=PROXYJOBSUBMISSION`
	To allow job submission, in the Torque configuration file (torque.cfg):
	`VALIDATEPATH FALSE`
	Run the following qmgr directive:
	`set server disable_server_id_check = True`
	Restart both Moab and pbs_server.

PARALLOCATIONPOLICY	
Format	One of `BestFit`, `BestFitP`, `FirstStart`, `LoadBalance`, `LoadBalanceP`, `Random`, or `RoundRobin`.
Default	`FirstStart`
Description	The approach to use to allocate resources when more than one eligible partition can be found. See 23.10 Grid Scheduling Policies for more information.
Example	`PARALLOCATIONPOLICY  LOADBALANCE` *New jobs will be started on the most lightly allocated partition.*

PARCFG	
Format	NODEPOWEROFFDURATION, NODEPOWERONDURATION, NODEALLOCATIONPOLICY or one or more key-value pairs as described in 6.2.3.B Per Job Resource Limits
Default	---
Description	The attributes, policies, and constraints for the given partition.
Example	`PARCFG[oldcluster] MAX.WCLIMIT=12:00:00` *Moab will not allow jobs to run on the `oldcluster` partition, which has a wallclock limit in excess of `12 hours`.*

PBSACCOUNTINGDIR	
Format	`<PATH>`
Default	---
Description	When specified, Moab will write out job events in standard PBS/ Torque tracejob format to the specified directory using the standard PBS/TORQUE log file naming convention. See 'Using tracejob to Locate Job Failures' in the *Torque Resource Manager Administrator Guide* for more information.
Example	`PBSACCOUNTINGDIR /var/spool/torque/sched_logs/` *Job events will be written to the specified directory (can be consumed by PBS's tracejob command).*

PERPARTITIONSCHEDULING	
Format	`<BOOLEAN>`
Default	`FALSE`

Description	By default, Moab's scheduling routine schedules each job on each partition using the following algorithm:
	prioritize
	foreach (job)
	    find the partition on which that job should run
	    schedule job
	In this model, a job's priority is the same on each partition as it uses a single global priority. Because a job's priority is the same on every partition, Moab prioritizes the queue once and then schedules the prioritized queue across all partitions.
	When `PERPARTITIONSCHEDULING TRUE` is set, the following algorithm is used:
	foreach (partition)
	    prioritize
	    foreach (job)
	        schedule job
	In this case, each partition can have a unique priority configuration and Moab will re-prioritize the jobs for each partition on the system. Each job is prioritized and scheduled on each partition. See the parameter PARCFG for more information. Also, note that Moab will order the partitions as they are discovered in the moab.cfg file. Partitions should be explicitly ordered via PARCFG in the moab.cfg file.

Example	`PERPARTITIONSCHEDULING TRUE`
	`PARCFG[p1] CONFIGFILE=/opt/moab/etc/p1.cfg`
	`PARCFG[p2] CONFIGFILE=/opt/moab/etc/p2.cfg`
	Rather than prioritizing the job queue once, Moab prioritizes the job queue for each partition, p1 and p2 respectively, and schedules each partition in turn using the policies located in their respective configuration files. See 6.2.5 Per-Partition Settings for more information.

PEWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The coefficient to be multiplied by a job's PE (processor equivalent) priority factor.

Example	`RESWEIGHT 10`
	`PEWEIGHT  100`
	*Each job's priority will be increased by `10` * `100` * its PE factor.*

PREEMPTIONALGORITHM	
Format	PREEMPTORCENTRIC or PREEMPTEECENTRIC
Default	`PREEMPTEECENTRIC`
Description	PREEMPTEECENTRIC specifies Moab will use a custom scheduling policy that ignores many policies such as JOBNODEMATCHPOLICY, NODEALLOCATIONPOLICY, NODEACCESSPOLICY. This custom scheduling policy will ensure the fewest and least important (by priority) preemptees are disturbed by the preemptor. PREEMPTORCENTRIC specifies Moab uses the normal scheduling policy and obeys all configured policies.
Example	`PREEMPTIONALGORITHM PREEMPTORCENTRIC` *Moab schedules the jobs as if the preemptees were not active and results in optimal placement for the preemptor.*

PREEMPTPOLICY	
Format	One of the following: CANCEL, REQUEUE, SUSPEND, or CHECKPOINT
Default	`REQUEUE`
Description	Specifies how preemptable jobs will be preempted: • If this policy is set to `REQUEUE`, preemptible jobs should be marked as RESTARTABLE. • If this policy is set to `SUSPEND`, preemptible jobs should be marked as SUSPENDABLE. Moab uses preemption escalation to preempt resources if the specified preemption facility is not applicable. This means if the policy is set to `SUSPEND` and the job is not SUSPENDABLE, Moab might attempt to requeue or even cancel the job.
Example	`PREEMPTPOLICY CHECKPOINT` *Jobs that are to be preempted will be checkpointed and restarted at a later time.*

PREEMPTPRIOJOBSELECTWEIGHT	
Format	`<DOUBLE>`
Default	`256.0`
Description	Determines which jobs to preempt based on size or priority. The higher the value, the more emphasis is placed on the priority of the job, causing the lower priority jobs to be preempted first. The lower the value, the more emphasis is placed on the size of the job, causing the smaller jobs to be preempted first. If set to `0`, job priority will be ignored, job size will take precedence and the smallest jobs will be preempted. The special setting of `-1` places the emphasis solely on resource utilization. This means that jobs will be preempted in a manner that keeps the resource utilization at the highest level, regardless of job priority or size.
Example	`PREEMPTPRIOJOBSELECTWEIGHT 220.5`

PREEMPTRTIMEWEIGHT	
Format	`<DOUBLE>`
Default	`0`
Description	If set to anything other than `0`, a job's remaining time is added into the calculation of which jobs will be preempted. If a positive weight is specified, jobs with a longer remaining time are favored. If a negative weight is specified, jobs with a shorter remaining time are favored.
Example	`PREEMPTRTIMEWEIGHT 1.5`

PREEMPTSEARCHDEPTH	
Format	`<INTEGER>`
Default	`unlimited`
Description	Specifies how many preemptible jobs will be evaluated as potential targets for serial job preemptors. See Chapter 21: Preemption for more information.
Example	`PREEMPTSEARCHDEPTH 8` *Serial job preemptors will only consider the first 8 feasible preemptee jobs when determining the best action to take.*

PRIORITYTARGETDURATION	
Format	`[[[DD:]HH:]MM:]SS`
Default	---
Description	The *ideal* job duration that will maximize the value of the WALLTIMEWEIGHT priority factor. If specified, this factor will be calculated as the distance from the ideal. Consequently, in most cases, the associated subcomponent weight should be set to a negative value.

Example	`WALLTIMEWEIGHT          -2500`
	`PRIORITYTARGETDURATION  1:00:00`

PRIORITYTARGETPROCCOUNT	
Format	`<INTEGER>`{+|-|%}
Default	---
Description	The ideal job requested proc count that will maximize the value of the PROCWEIGHT priority factor. If specified, this factor will be calculated as the distance from the ideal (proc count - ideal = coefficient of PROCWEIGHT). Consequently, in most cases, the associated subcomponent weight should be set to a negative value.

Example	`PROCWEIGHT              -1000`
	`PRIORITYTARGETPROCCOUNT  64`

PROCWEIGHT	
Format	`<INTEGER>`
Default	0
Description	The coefficient to be multiplied by a job's requested processor count priority factor.
Example	`PROCWEIGHT 2500`

PROFILECOUNT	
Format	`<INTEGER>`
Default	600
Description	The number of statistical profiles to maintain. PROFILECOUNT must be set high enough that at least one day of statistics is maintained. The statistics time window can be determined by measuring PROFILEDURATION * PROFILECOUNT. If PROFILEDURATION is one hour, then PROFILECOUNT must be at least 24, so 24 hours' worth of statistics are maintained. If PROFILEDURATION is 30:00 then PROFILECOUNT must be set to at least 48. If PROFILECOUNT is not high enough for at least one day of statistics, Moab adjusts it automatically.
Example	`PROFILECOUNT 300`

PROFILEDURATION	
Format	`[[[DD:]HH:]MM:]SS`
Default	`00:30:00`
Description	The duration of each statistical profile. The duration cannot be more than 24 hours, and any specified duration must be a factor of 24. For example, factors of 1/4, 1/2, 1, 2, 3, 4, 6, 8, 12, and 24 are acceptable durations.
Example	`PROFILEDURATION 24:00:00`

PURGETIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`0`
Description	The amount of time Moab will keep a job or node record for an object no longer reported by the resource manager. Useful when using a resource manager that 'drops' information about a node or job due to internal failures. **Note**: This parameter is superseded by JOBPURGETIME.
Example	`PURGETIME 00:05:00` *Moab will maintain a job or node record for `5 minutes` after the last update regarding that object received from the resource manager.*

PUSHCACHETOWEBSERVICE	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether or not you want to send cache objects (nodes, jobs, services, etc.) to Moab Web Services.
Example	`PUSHCACHETOWEBSERVICE TRUE`

QOSCFG[<QOSID>]

Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: General Credential Flags, ACLBLTHRESHOLD, ACLQTTHRESHOLD, ACLXFTHRESHOLD, ENABLEPROFILING, FSTARGET, JOBPRIOACCRUALPOLICY, JOBPRIOEXCEPTIONS, MEMBERULIST, PLIST, PREEMPTEES, PREEMPTMAXTIME, PREEMPTMINTIME, PREEMPTQTTHRESHOLD, PREEMPTXFTHRESHOLD, PRIORITY, QFLAGS, QTTARGET, QTWEIGHT, REQRID, RSVQTTHRESHOLD, RSVXFTHRESHOLD, XFTARGET, XFWEIGHT, usage limit.
Default	---
Description	Specifies QOS specific attributes. See 2.8 Job Flags for a description of valid flag values. See 6.3 Quality of Service (QoS) Facilities for more information.
Example	`QOSCFG[commercial] PRIORITY=1000 MAXJOB=4 MAXPROC=80` *Moab will increase the priority of jobs using QOS commercial, and will allow up to 4 simultaneous QOS commercial jobs with up to 80 total allocated processors.*

QOSDEFAULTORDER

Format	Comma-delimited list of QOS names.
Default	---
Description	Sets a global QOS default order for all QOSs, which overrides any specific default QOS. If the order is defined as `b,a,c` and a user has access to `c,a` and submits a job without requesting a specific QOS, the job is assigned `a` as the default QOS.
Example	`QOSDEFAULTORDER  b,a,c` *If the job does not have a QOS specified, it is assigned a QOS from the QOSDEFAULTORDER list (if the user has access to one of them).*

QOSISOPTIONAL	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	An entity's default QOS will be the first QOS specified in the QLIST parameter. When this parameter is set to `TRUE` the default QOS for the associated credential (user, account, class, etc.) will not be automatically set to the first QOS specified in the QLIST.

Example	`QOSISOPTIONAL TRUE`
	`USERCFG[bob]  QLIST=high,low`
	Moab will set the QOSList for user `bob` to `high` and `low` but will not set the QDEF. Should `bob` decide to submit to a particular QOS he will have to do so manually.

QOSREJECTPOLICY	
Format	One or more of `CANCEL`, `HOLD`, `IGNORE`, or `MAIL`.
Default	`HOLD`
Description	The action to take when Moab determines that a job cannot access a requested QoS. `CANCEL` issues a call to the resource manager to cancel the job. `HOLD` places a *batch* hold on the job preventing the job from being further evaluated until released by an admin. (**Note**: Admins can dynamically alter job attributes and possibly *fix* the job with mjobctl -m.) With `IGNORE`, Moab will ignore the QoS request and schedule the job using the default QoS for that job. MAIL will send email to both the admin and the user when QoS request violations are detected. Most combinations of attributes can be specified; however, if both `MAIL` and `IGNORE` are specified, Moab will not implement `MAIL`. Similarly, while `CANCEL` and `HOLD` are mutually exclusive, `CANCEL` will supersede `HOLD` if both are specified. Also see the parameter JOBREJECTPOLICY.
Example	`QOSREJECTPOLICY  MAIL,CANCEL`

QOSWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The weight to be applied to the qos priority of each job (see 4.1.2.B Credential (CRED) Component).
Example	`QOSWEIGHT 10`

QUEUETIMECAP	
Format	`<DOUBLE>`
Default	`0` (NO CAP)
Description	The maximum allowed absolute preweighted queuetime priority factor.

Example	`QUEUETIMECAP    10000`
	`QUEUETIMEWEIGHT 10`
	*A job that has been queued for 40 minutes will have its queuetime priority factor calculated as 'Priority = QUEUETIMEWEIGHT * MIN (10000,40)'.*

QUEUETIMEWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	Specifies multiplier applied to a job's queue time (in minutes) to determine the job's queuetime priority factor.
Example	`QUEUETIMEWEIGHT 20` *A job that has been queued for 4:20:00 will have a queuetime priority factor of `20` * 260.*

REALTIMEDBOBJECTS	
Format	Comma-delimited list of one or more of the following: `JOB`, `NODE`, `RSV` (reservation), `TRIG` (trigger), `VC` (virtual container). You can also specify `ALL` or `NONE`.
Default	`ALL`
Description	Specifies which objects Moab will store in the unixodbc database.
Example	`REALTIMEDBOBJECTS JOB,RSV,TRIG` *Moab stores jobs, reservations, and triggers in the unixodbc database. It will no longer record real time information about nodes and VCs.*

RECORDEVENTLIST	
Format	One or more comma (',') or plus ('+') separated events of GEVENT, ALLSCHEDCOMMAND, AMCREATE, AMDELETE, AMEND, AMPAUSE, AMQUOTE, AMRESUME, AMSTART, AMUPDATE, JOBCANCEL, JOBCHECKPOINT, JOBEND, JOBFAILURE, JOBHOLD, JOBMIGRATE, JOBMODIFY, JOBPREEMPT, JOBREJECT, JOBRESUME, JOBSTART, JOBSUBMIT, JOBVARSET, JOBVARUNSET, NODEADD, NODEDELETE, NODEDESTROY, NODEDOWN, NODEFAILURE, NODEMODIFY, NODEPOWEROFF, NODEPOWERON, NODEPROVISION, NODEUP, NOTE, QOSVIOLATION, RMDOWN, RMPOLLEND, RMPOLLSTART, RMUP, RSVCANCEL, RSVCREATE, RSVEND, RSVMODIFY, RSVSTART, SCHEDCOMMAND, SCHEDCYCLEEND, SCHEDCYCLESTART, SCHEDFAILURE, SCHEDMODIFY, SCHEDPAUSE, SCHEDRECYCLE, SCHEDRESUME, SCHEDSTART, SCHEDSTOP, TRIGEND, TRIGFAILURE, TRIGSTART, TRIGTHRESHOLD, or ALL
Default	JOBSTART, JOBCANCEL, JOBEND, JOBFAILURE, SCHEDPAUSE, SCHEDSTART, SCHEDSTOP, TRIGEND, TRIGFAILURE, TRIGSTART
Description	Specifies which events should be recorded in the appropriate event file found in Moab's `stats/` directory. These events are recorded for both local and remotely staged jobs (see 12.2.4 Event Logs). **Note**: If a plus character is included in the list, the specified events will be added to the default list; otherwise, the specified list will replace the default list.
Example	`RECORDEVENTLIST JOBSTART,JOBCANCEL,JOBEND` *When a local and/or remote job starts, is canceled, or ends, the respective event will be recorded.*

REJECTDOSSCRIPTS	
Format	<BOOLEAN>
Default	TRUE
Description	Moab rejects DOS-formatted scripts submitted with the *msub* command. This is useful if you use Slurm as your resource manager, since it does not handle DOS scripts well. For REJECTDOSSCRIPTS to work correctly, FILTERCMDFILE must be FALSE. Otherwise, Moab modifies the script instead of rejecting it, leading to job errors. This parameter is deprecated and may be removed in a future release.
Example	`REJECTDOSSCRIPTS FALSE`

REJECTDOSSCRIPTS

	Moab does not reject DOS-formatted scripts submitted with `msub`.

REJECTINFEASIBLEJOBS

Format	`<BOOLEAN>`
Default	`FALSE`
Description	If zero feasible nodes are found for a job among all the nodes on the cluster and all the resource managers are reporting 'Active', the scheduler rejects the job. See the parameter JOBREJECTPOLICY for more information.

Example	`REJECTINFEASIBLEJOBS TRUE`
	`JOBREJECTPOLICY      MAIL,CANCEL`
	Any job with zero feasible nodes for execution will be rejected.

REJECTNEGPRIOJOBS

Format	`<BOOLEAN>`
Default	`TRUE`
Description	If enabled, the scheduler will refuse to start any job with a negative priority. See 4.1.1 Job Priority Overview and the parameter ENABLENEGJOBPRIORITY for more information.

Example	`ENABLENEGJOBPRIORITY TRUE`
	`REJECTNEGPRIOJOBS    TRUE`
	Any job with a priority less than zero will be rejected.

REMAPCLASS

Format	<ClassID>
Default	---
Description	Specifies which class/queue will be remapped based on the processors, nodes, and node features requested and the resource limits of each class. See 2.7.5.H Creating a Remap Class for more information. In order to use `REMAPCLASS`, you must specify a `DEFAULTCLASS`.

Example	`RMCFG[internal]  DEFAULTCLASS=batch`
	`REMAPCLASS       batch`
	`CLASSCFG[small]  MAX.PROC=2`
	`CLASSCFG[medium] MAX.PROC=16`
	`CLASSCFG[large]  MAX.PROC=1024`
	Class `batch` will be remapped based on the number of processors requested.

REMAPCLASSLIST

Format	Comma-delimited list of class names.
Default	---
Description	The order in which classes will be searched when attempting to remap a class. Only classes included in the list will be searched and Moab will select the first class with matches. **Note**: If no `REMAPCLASSLIST` is specified, Moab will search all classes and will search them in the order they are discovered. See 2.7.5.H Creating a Remap Class for more information.

Example	`RMCFG[internal] DEFAULTCLASS=batch`
	`REMAPCLASS      batch`
	`REMAPCLASSLIST  short,medium,long`
	Class `batch` will be re-mapped to one of the listed classes.

REMOTEFAILTRANSIENT	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Only applicable to Moab configurations with multiple resource managers able to run jobs (such as in a grid environment). When Moab attempts to migrate a job to one of these resource managers, a remote failure may occur. For example, a destination peer in a grid that has an error accepting a job results in a remote error, and the job is rejected. `REMOTEFAILTRANSIENT` controls how Moab reacts to remote errors. By default, Moab considers such an error permanent and does not try to migrate the same job to that resource manager again. If `REMOTEFAILTRANSIENT` is set to `TRUE`, then Moab considers such an error as transient and will not exclude the erring resource manager in future migration attempts.
Example	`REMOTEFAILTRANSIENT     TRUE`

REMOVETRIGOUTPUTFILES	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	When Moab launches external trigger actions, the standard output and error of those trigger actions are redirected to files located in Moab's spool directory. By default, these files are cleaned every 24 hours. (Files older than 24 hours are removed.) If, however, you want to have Moab immediately remove the spool files after they are no longer needed, set RemoveTrigOutputFiles to `TRUE`.
Example	`REMOVETRIGOUTPUTFILES   TRUE`

RESCAP	
Format	`<DOUBLE>`
Default	`0` (NO CAP)
Description	The maximum allowed absolute preweighted job resource priority factor.
Example	`RESCAP 1000` *The total resource priority factor component of a job will be bound by +/-* `1000`

RESERVATIONDEPTH[X]	
Format	`<INTEGER>`
Default	`1`
Description	The number of priority reservations that are allowed in the associated reservation bucket. **Note**: The array index, *X*, is the bucket label and can be any string up to 64 characters. This label should be synchronized with the RESERVATIONQOSLIST parameter. See 6.1.4 Reservation Policies.

Example	`RESERVATIONDEPTH[bigmem]    4`
	`RESERVATIONQOSLIST[bigmem] special,fast,joshua`
	Jobs with QOSs of `special`*,* `fast`*, or* `joshua` *can have a cumulative total of up to 4 priority reservations.*

RESERVATIONPOLICY	
Format	One of the following: `CURRENTHIGHEST`, `HIGHEST`, `NEVER`.
Default	`CURRENTHIGHEST`
Description	Specifies how Moab reservations will be handled. See also the parameter RESERVATIONDEPTH. See 6.1.4 Reservation Policies.

Example	`RESERVATIONPOLICY          CURRENTHIGHEST`
	`RESERVATIONDEPTH[DEFAULT]  2`
	Moab will maintain reservations for only the `2` *currently highest priority jobs.*

RESERVATIONQOSLIST[X]	
Format	One or more QOS values or `[ALL]`.
Default	`[ALL]`
Description	Specifies which QOS credentials have access to the associated reservation bucket. **Note**: The array index, *X*, is the bucket label and can be any string up to 64 characters. This label should be synchronized with the RESERVATIONDEPTH parameter. See 6.1.4 Reservation Policies.

Example	`RESERVATIONDEPTH[big]    4`
	`RESERVATIONQOSLIST[big] hi,low,med`
	Jobs with QOSs of `hi`, `low`, or `med` can have a cumulative total of up to `4` priority reservations.

RESERVATIONRETRYTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`60 seconds`
Description	Period of time Moab will continue to attempt to allocate resources to start a job after the time resources should be made available. This parameter takes into account resource manager node state race conditions, nodes with residual high load, network glitches, etc. For related information, see 6.1.4 Reservation Policies, and the parameters DEFERSTARTCOUNT, DEFERTIME, NODEFAILURERESERVETIME, JOBRETRYTIME, and GUARANTEEDPREEMPTION.
Example	`RESERVATIONRETRYTIME    00:05:00` *Moab will try for up to `5 minutes` to maintain immediate reservations if the reservations are blocked due to node state, network, or batch system based race conditions.*

RESOURCELIMITMULTIPLIER[<PARID>]	
Format	`<RESOURCE>:<MULTIPLIER>[,...]` Where `<RESOURCE>` is one of the following: `NODE`, `PROC`, `JOBPROC`, `MEM`, `JOBMEM`, `SWAP`, `DISK`, or `WALLTIME`.
Default	`1.0`
Description	If set to less than one, then the hard limit will be the specified limit and the soft limit will be the specified limit multiplied by the multiplier. If set to a value greater than one, then the specified limit will be the soft limit and the hard limit will be the specified limit multiplied by the multiplier. See 5.2.6 Usage-Based Limits.
Example	`RESOURCELIMITMULTIPLER  PROC:1.1,MEM:2.0` *Sets hard limit for `PROC` at `1.1` times the `PROC` soft limit, and the hard limit of `MEM` to `2.0` times the `MEM` soft limit.*

RESOURCELIMITPOLICY	
Format	`<RESOURCE>:[<SPOLICY>,]<HPOLICY> : [<SACTION>,]<HACTION> [: [<SVIOLATIONTIME>,]<HVIOLATIONTIME>]...` Where`RESOURCE` is one of `CPUTIME`, `DISK`, `JOBMEM`, `JOBPROC`, `MEM`, `MINJOBPROC`, `NETWORK`, `PROC`, `SWAP`, or `WALLTIME`. Where`*POLICY` is one of `ALWAYS`, `EXTENDEDVIOLATION`, or `BLOCKEDWORKLOADONLY` and where `*ACTION` is one of `CANCEL`, `CHECKPOINT`, `NOTIFY`, `REQUEUE`, `SIGNAL`, or `SUSPEND`.
Default	No limit enforcement.
Description	Specifies how the scheduler should handle jobs that utilize more resources than they request. See 5.2.6 Usage-Based Limits.
Example	`RESOURCELIMITPOLICY  MEM:ALWAYS,BLOCKEDWORKLOADONLY:REQUEUE,CANCEL` *Moab will cancel all jobs that exceed their requested memory limits.*

RESTARTINTERVAL	
Format	`[[[DD:]HH:]MM:]SS`
Default	---
Description	Causes Moab daemon to recycle/restart when the given interval of time has transpired.
Example	`RESTARTINTERVAL  20:00:00` *Moab daemon will automatically restart every `20 hours`.*

RESOURCEQUERYDEPTH	
Format	`<INTEGER>`
Default	3
Description	Maximum number of options that will be returned in response to an mshow -a resource query.
Example	`RESOURCEQUERYDEPTH  1` *The `mshow -a` command will return at most `1` valid collection of resources.*

RESWEIGHT	
Format	`<INTEGER>`
Default	1
Description	All resource priority components are multiplied by this value before being added to the total job priority. See 4.1 Job Prioritization.

Example	`RESWEIGHT  5`
	`MEMWEIGHT  10`
	`PROCWEIGHT 100`
	`SWAPWEIGHT 0`
	`RESCAP     2000`
	*The job priority resource factor will be calculated as MIN (`2000,5` * (`10` * JobMemory + `100` * JobProc)).*

RMCFG

Format	One or more key-value pairs as described in 11.2 Resource Manager Configuration.
Default	---
Description	The interface and policy configuration for the scheduler-resource manager interface. Described in detail in 11.2 Resource Manager Configuration.
Example	`RMCFG[Torque3] TYPE=PBS` *The PBS server will be used for resource management.*

RMMSGIGNORE

Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether or not Moab should adjust node state based on generic resource manager failure messages. See 'Compute Node Health Check' in the *Torque Resource Manager Administrator Guide* for more information. For green or ONDEMAND computing, RMMSGIGNORE must be set to `TRUE` to prevent Moab from powering off a down node.
Example	`RMMSGIGNORE TRUE` *Moab will load and report resource manager failure messages but will not adjust node state as a result of them.*

RMPOLLINTERVAL	
Format	`[<MINPOLLTIME>,]<MAXPOLLTIME>` Where poll time is specified as `[[[DD:]HH:]MM:]SS`.
Default	`0,30`
Description	The interval between resource manager polls. The poll interval will be no less than MINPOLLTIME and no more than MAXPOLLTIME. If you specify a single value, Moab interprets the value as the MAXPOLLTIME with a MINPOLLTIME of 0. If you use Torque as your resource manager, prevent communication errors by giving tcp_timeout at least twice the value of the Moab `RMPOLLINTERVAL`.
Example	`RMPOLLINTERVAL 30,45` *Moab will refresh its resource manager information between a minimum of `30 seconds` and a maximum of `45 seconds`.* ***Note****: This parameter specifies the default global poll interval for all resource managers.*

RMRETRYTIMECAP	
Format	`[[[DD:]HH:]MM:]SS`
Default	`1:00:00`
Description	Moab attempts to contact RMs that are in state 'corrupt' (not down). If the attempt is unsuccessful, Moab tries again later. If the second attempt is unsuccessful, Moab increases the gap (the gap grows exponentially) between communication attempts. `RMRETRYTIMECAP` puts a cap on the length between connection attempts.
Example	`RMRETRYTIMECAP 24:00:00` *Moab stops increasing the gap between connection attempts once the retry gap reaches `24 hours`.*

RSVLIMITPOLICY	
Format	`HARD` or `SOFT`
Default	---
Description	Specifies what limits should be enforced when creating reservations.
Example	`RSVLIMITPOLICY  HARD` *Moab will limit reservation creation based on the `HARD` limits configured.*

RSVNODEALLOCATIONPOLICY	
Format	One of the following: FIRSTAVAILABLE, LASTAVAILABLE, MINRESOURCE, CPULOAD, CONTIGUOUS, MAXBALANCE, or PRIORITY
Default	`LASTAVAILABLE`
Description	Specifies how Moab should allocate available resources to reservations.
Example	`RSVNODEALLOCATIONPOLICY MINRESOURCE` *Moab will apply the node allocation policy `MINRESOURCE` to all reservations by default.*

RSVNODEALLOCATIONPRIORITYF	
Format	User specified algorithm.
Default	---
Description	When RSVNODEALLOCATIONPOLICY is set to `PRIORITY`, this parameter enables you to specify your own priority algorithm. The priority functions available are the same as the node priority functions.

Example	`RSVNODEALLOCATIONPOLICY PRIORITY`
	`RSVNODEALLOCATIONPRIORITYF 'SPEED + .01 * AMEM - 10 * JOBCOUNT'`

RSVPROFILE[X]	
Format	One or more of the following: **Allowed:** TRIGGERACL (ACCOUNTLIST, CLASSLIST, GROUPLIST, MAXTIME, QOSLIST, USERLIST) HostExp (HOSTLIST) Features (NODEFEATURES) FLAGS TASKCOUNT RSVACCESSLIST **Note**: Lists of more than one ACL value cannot be whitespace delimited. Such lists must be delimited with a comma, pipe, or colon. **Not allowed:** ACCESS CHARGEACCOUNT DAYS DEPTH ENDTIME OWNER PARTITION PERIOD PRIORITY RESOURCES STARTTIME TPN
Default	---
Description	Specifies attributes of a reservation profile using syntax similar to that for specifying a standing reservation. See 6.1.2.B Using Reservation Profiles for details.

Example	`RSVPROFILE[fast] USERLIST=john,steve`
	`RSVPROFILE[fast] QOSLIST=high,low`
	`RSVPROFILE[fast] TRIGGER=ETYPE=start,OFFSET=5:00,ATYPE=exec, ACTION="/opt/moab/rp.pl"`
	Moab will create a reservation profile including trigger and ACL information.

RSVSEARCHALGO	
Format	LONG or WIDE
Default	NONE
Description	When Moab is determining when and where a job can run, it either searches for the most resources (WIDE) or the longest range of resources (LONG). In almost all cases, searching for the longest range is ideal and returns the soonest starttime. In some rare cases, however, a particular job may need to search for the most resources. In those cases sites can configure this parameter to prevent the starvation of large jobs that fail to hold onto their reservation starttimes. See the WIDERSVSEARCHALGO job flag. If this parameter is not set, it will be displayed in *mschedctl -l* as NONE but the algorithm that is used will be LONG.
Example	`RSVSEARCHALGO WIDE`

SCHEDCFG	
Format	List of zero or more space delimited <ATTR>=<VALUE> pairs. Where <ATTR> is one of the following: FBSERVER, FLAGS, MAXJOBID, MINJOBID, HTTPSERVERPORT, MODE, RECOVERYACTION, SERVER, TRIGGER, or USEDATABASE MAXRECORDEDCJOBID is deprecated.
Default	---
Description	Specifies scheduler policy and interface configuration. The SERVER attribute can also be set using the environment variable $MOABSERVER. Using this variable enables you to quickly change the Moab server that client commands will connect to. `> export MOABSERVER=cluster2:12221`
Example	`SCHEDCFG[zylem3] SERVER=geronimo.scc.com:3422 MODE=NORMAL` *Moab will execute in NORMAL mode on the host geronimo.scc.com.*

SERVERCSALGO	
Format	HMAC64|HMACSHA2
Default	`HMAC64`
Description	Sets the algorithm used for message digests and message authentication codes: • HMAC64: the default (SHA-1) • HMACSHA2: more secure (SHA-512) If you are using Moab Web Services, then you must set the MWS configuration parameter moab.messageDigestAlgorithm to match the value of SERVERCSALGO. See 'moab.messageDigestAlgorithm' in the *Moab Web Services Administrator Guide* for more information.
Example	`SERVERCSALGO HMACSHA2` *Moab will use SHA-512 for message digests and message authentication codes.*

SERVERHOST	
Description	This parameter is deprecated and may be removed in a future release. See the parameter SCHEDCFG for replacement parameter.

SERVERMODE	
Description	This parameter is deprecated and may be removed in a future release. See the parameter SCHEDCFG for replacement parameter.

SERVERNAME

Format	`<STRING>`
Default	`<SERVERHOST>`
Description	The name the scheduler will use to refer to itself in communication with peer daemons. See the parameter SCHEDCFG for replacement parameter.
Example	`SERVERNAME moabA`

SERVERPORT

Format	`<INTEGER>` (range: 1-64000)
Default	`40559`
Description	Port on which moab will open its user interface socket. See the parameter SCHEDCFG for replacement parameter.
Example	`SERVERPORT 30003` *Moab will listen for client socket connections on port `30003`.*

SERVERSUBMITFILTER

Format	`<PATH>`
Default	---
Description	The location of a global job submit filter script. When you configure a global job submit filter, Moab executes it on the head node and uses it to filter every job submission it receives. See the section Server-Based Submit Filter for more information about job submit filters.
Example	`SERVERSUBMITFILTER /opt/moab/scripts/globalfilter.pl` *Moab uses `/opt/moab/scripts/globalfilter.pl` to filter every job submitted to Moab.*

SERVICEWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The service component weight associated with the service factors. See 4.1.2.E Service (SERVICE) Component for more information.
Example	`SERVICEWEIGHT 2`

SHOWMIGRATEDJOBSASIDLE	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	By default, migrated jobs in the grid will show as blocked. This is to prevent jobs from counting against the idle policies of multiple clusters rather than just the cluster to which the job was migrated.
Example	`SHOWMIGRATEDJOBSASIDLE TRUE` *When set to `TRUE`, migrated jobs will show as idle and will count against the idle policies of the cluster showing the job as migrated.*

SPOOLDIR	
Format	`<STRING>`
Default	---
Description	The directory for temporary spool files created by Moab while submitting a job to the resource manager.
Example	`SPOOLDIR /tmp/moab/spool`

SPOOLDIRKEEPTIME	
Format	`<INTEGER>` (seconds) or [[[DD:]HH:]MM:]SS
Default	---
Description	The interval to delete spool files and other temporary files that have been left in the spool directory. If not set, Moab will remove the spool files after a year.
Example	`SPOOLDIRKEEPTIME 4:00:00`

SPVIOLATIONWEIGHT	
Format	`<INTEGER>`
Default	0
Description	The weight to be applied to a job that violates soft usage limit policies (see 4.1.2.E Service (SERVICE) Component).
Example	`SPVIOLATIONWEIGHT 5000`

SRCFG[X]	
Format	One or more of the following `<ATTR>=<VALUE>` pairs: `ACCESS`, ACCOUNTLIST, CHARGE, CHARGEACCOUNT, CHARGEUSER, `CLASSLIST`, CLUSTERLIST, COMMENT, DAYS, `DEPTH`, DISABLE, ENDTIME, FLAGS, `GROUPLIST`, HOSTLIST, `JOBATTRLIST`, `MAXTIME`, `NODEFEATURES`, OWNER, `PARTITION`, PERIOD, PROFILE, `PRIORITY`, `QOSLIST`, REQUIREDACCTLIST, REQUIREDTPN, REQUIREDUSERLIST, RESOURCES, ROLLBACKOFFSET, RSVACCESSLIST, RSVGROUP, STARTTIME, TASKCOUNT, `TIMELIMIT`, `TPN`, `TRIGGER`, or USERLIST **Note:** `HOSTLIST` and `ACL` list values must be comma-delimited. For example: `HOSTLIST=nodeA,nodeB`
Default	---
Description	Specifies attributes of a standing reservation. See 6.1.5 Configuring and Managing Reservations for details.

Example	`SRCFG[fast] STARTTIME=9:00:00 ENDTIME=15:00:00`
	`SRCFG[fast] HOSTLIST=node0[1-4]$`
	`SRCFG[fast] QOSLIST=high,low`
	Moab will create a standing reservation running from `9:00 A.M.` to `3:00 P.M.` on nodes `1` through `4` accessible by jobs with QOS `high` or `low`.

STARTCOUNTCAP	
Format	`<INTEGER>`
Default	0
Description	The max weighted value allowed from the startcount subfactor when determining a job's priority (see 4.1.2 Job Priority Factors for more information).

Example	`STARTCOUNTWEIGHT 5000`
	`STARTCOUNTCAP    30000`

STARTCOUNTWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight to be applied to a job's startcount when determining a job's priority (see 4.1.2 Job Priority Factors for more information).
Example	`STARTCOUNTWEIGHT 5000`

STATDIR	
Format	`<STRING>`
Default	`stats`
Description	The directory where Moab statistics will be maintained.
Example	`STATDIR /var/adm/moab/stats`

STATPROCMAX

Format	`<INTEGER>`
Default	`1`
Description	The maximum number of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). **Caution:** Altering this setting will reset all recorded statistical data related to it. Do not change this parameter via *`mschedctl -m`* (or *`changeparam`*). Moab only reads in this setting when starting up (or restarting).

Example	
	```
	STATPROCMAX      256
	STATPROCSTEPCOUNT 4
	STATPROCSTEPSIZE  4
	```
	 Each matrix output will display data in rows for jobs requesting between `4` and `256` processors. A **NONE** in services will still allow users to run showq and checkjob on their own jobs.

STATPROCMIN	
Format	`<INTEGER>`
Default	`1`
Description	The minimum number of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). **Caution:** Altering this setting will reset all recorded statistical data related to it. Do not change this parameter via *`mschedctl -m`* (or *`changeparam`*). Moab only reads in this setting when starting up (or restarting).

Example	`STATPROCMIN        4`
	`STATPROCSTEPCOUNT  4`
	`STATPROCSTEPSIZE   4`
	
	Each matrix output will display data in rows for jobs requesting between `4` and 256 processors.
	
	A **NONE** in services will still allow users to run showq and checkjob on their own jobs.

STATPROCSTEPCOUNT	
Format	`<INTEGER>`
Default	`5`
Description	The number of rows of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). **Caution:** Altering this setting will reset all recorded statistical data related to it. Do not change this parameter via *`mschedctl -m`* (or *`changeparam`*). Moab only reads in this setting when starting up (or restarting).

Example	`STATPROCMIN       4`
	`STATPROCSTEPCOUNT 4`
	`STATPROCSTEPSIZE  4`
	
	Each matrix output will display data in rows for jobs requesting between `4` and 256 processors.

STATPROCSTEPSIZE	
Format	`<INTEGER>`
Default	`4`
Description	The processor count multiplier for rows of processors requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). **Caution:** Altering this setting will reset all recorded statistical data related to it. Do not change this parameter via *`mschedctl -m`* (or *`changeparam`*). Moab only reads in this setting when starting up (or restarting).

Example	`STATPROCMIN       4`
	`STATPROCSTEPCOUNT 4`
	`STATPROCSTEPSIZE  4`
	
	Each matrix output will display data in rows for jobs requesting between `4` and 256 processors.

STATTIMEMAX

Format	`[[DD:]HH:]MM:]SS`
Default	`00:15:00`
Description	The maximum amount of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). **Caution:** Altering this setting will reset all recorded statistical data related to it. Do not change this parameter via *mschedctl -m* (or *changeparam*). Moab only reads in this setting when starting up (or restarting).

Example	`STATTIMEMAX       02:08:00`
	`STATTIMESTEPCOUNT 4`
	`STATTIMESTEPSIZE  4`
	Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

STATTIMEMIN

Format	`[[DD:]HH:]MM:]SS`
Default	`00:15:00`
Description	The minimum amount of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). **Caution:** Altering this setting will reset all recorded statistical data related to it. Do not change this parameter via *mschedctl -m* (or *changeparam*). Moab only reads in this setting when starting up (or restarting).

Example	`STATTIMEMIN       00:02:00`
	`STATTIMESTEPCOUNT 4`
	`STATTIMESTEPSIZE  4`
	Each matrix output will display data in columns for jobs requesting between 2 and 128 minutes.

STATTIMESTEPCOUNT	
Format	`<INTEGER>`
Default	`6`
Description	The number of columns of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). **Caution:** Altering this setting will reset all recorded statistical data related to it. Do not change this parameter via *`mschedctl -m`* (or *`changeparam`*). Moab only reads in this setting when starting up (or restarting).

Example	`STATTIMEMIN 00:02:00`
	`STATTIMESTEPCOUNT 4`
	`STATTIMESTEPSIZE 4`
	Each matrix output will display data in columns for jobs requesting between `2` and `128` minutes.

STATTIMESTEPSIZE	
Format	`<INTEGER>`
Default	`4`
Description	The time multiplier for columns of time requested by jobs to be displayed in matrix outputs (as displayed by the showstats -f command). **Caution:** Altering this setting will reset all recorded statistical data related to it. Do not change this parameter via *`mschedctl -m`* (or *`changeparam`*). Moab only reads in this setting when starting up (or restarting).

Example	`STATTIMEMIN 00:02:00`
	`STATTIMESTEPCOUNT 4`
	`STATTIMESTEPSIZE 4`
	Each matrix output will display data in columns for jobs requesting between `2` and `128` minutes.

STOPITERATION	
Format	`<INTEGER>`
Default	`-1` (don't stop)
Description	Specifies which scheduling iteration Moab will stop and wait for a command to resume scheduling.
Example	`STOPITERATION 10` *Moab should stop after iteration `10` of scheduling and wait for administrator commands.*

STOREJOBSUBMISSION	
Format	`<BOOLEAN>`
Default	---
Description	When set to `TRUE`, specifies that Moab will save a job's submit arguments and script to `$MOABHOMEDIR/stats/jobarchive/jobNumber`. `STOREJOBSUBMISSION` currently does not work with jobs submitted using the Torque qsub command. Instead use Torque job logging. For more information, see 'Job Logging' in the *Torque Resource Manager Administrator Guide*. Moab does not manage any of the files it creates in the stats directory. Therefore, cluster administrators should keep this fact in mind when enabling `STOREJOBSUBMISSION`, and implement appropriate archival and/or pruning tasks to avoid overuse of disk space.
Example	`STOREJOBSUBMISSION TRUE`

STRICTPROTOCOLCHECK	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies how Moab reacts to differences in XML protocols when communicating with other Moab peers. If set to `TRUE`, Moab will reject any communication that does not strictly conform to the expected protocol. If set to `FALSE` (the default), Moab will not reject XML that has extra or unknown attributes.
Example	`STRICTPROTOCOLCHECK TRUE` *Moab will reject any XML communication that does not strictly conform to the expected protocol definition.*

STRICTSCHEDULING	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	If Moab fails to get a reservation or an allocation for a priority job, then it will stop the scheduling cycle for that iteration and not attempt to start any other jobs, even through backfill.
Example	`STRICTSCHEDULING TRUE` *Moab will stop the scheduling cycle for one iteration if it fails to get a reservation or allocation for a priority job. It will not attempt to start any other jobs until the next iteration.*

SUBMITENVFILELOCATION	
Format	`FILE` or `PIPE`
Default	---
Description	If set to `FILE`, these behaviors are expected: • The environment file is owned by a user with 600 permissions. • Moab writes the environment variables ('\0' delimited) to a random file in Moab's spool directory. • Moab adds the `--export-file=<path_to_file>` on the sbatch command line. • Moab deletes the file after the job completes. If set to `PIPE`, these behaviors are expected: • Moab creates a pipe and passes the read end of the pipe's file descriptor to sbatch. • Moab's parent process writes the environment ('\0' delimited) into the write end of the pipe. We recommend that you configure this parameter, for a more secure environment.
Example	`SUBMITENVFILELOCATION PIPE`

SUBMITFILTER	
Format	`<STRING>`
Default	---
Description	The directory of a given submit filter script.
Example	`SUBMITFILTER /home/submitfilter/filter.pl`

SUBMITHOSTS	
Format	Space delimited list of host names.
Default	---
Description	If specified, `SUBMITHOSTS` specifies an explicit list of hosts where jobs can be submitted.
Example	`SUBMITHOSTS hostA hostB`

SUSPENDRESOURCES[<PARID>]	
Format	`<RESOURCE>[,...]` Where `<RESOURCE>` is one of the following: `NODE`, `PROC`, `MEM`, `SWAP`, or `DISK`
Default	---
Description	List of resources to dedicate while a job is suspended.
Example	`SUSPENDRESOURCES[base]  MEM,SWAP,DISK` *While a job is suspended in partition `base`, the memory, swap and disk for that job will remain dedicated to the job.*

SWAPWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The priority weight assigned to the virtual memory request of a job.
Example	`SWAPWEIGHT 10`

SYSCFG	
Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: `PRIORITY`, `FSTARGET`, `QLIST`, `QDEF`, `PLIST`, `FLAGS`, or a Fairness specification.
Default	---
Description	Specifies system-wide default attributes. See 2.8 Job Flags for more information.
Example	`SYSCFG PLIST=Partition1 QDEF=highprio` *By default, all jobs will have access to partition `Partition1` and will use the QOS `highprio`.*

SYSTEMMAXPROCPERJOB	
Format	`<INTEGER>`
Default	`-1` (NO LIMIT)
Description	The maximum number of processors that can be requested by any single job.
Example	`SYSTEMMAXPROCPERJOB 256` *Moab will reject jobs requesting more than `256` processors.*

SYSTEMMAXPROCSECONDPERJOB	
Format	`<INTEGER>`
Default	`-1` (NO LIMIT)
Description	The maximum number of proc-seconds that can be requested by any single job.
Example	`SYSTEMMAXJOBPROCSECOND 86400` *Moab will reject jobs requesting more than `86400` procs seconds (i.e., 64 processors * 30 minutes will be rejected, while a 2 processor * 12 hour job will be allowed to run).*

SYSTEMMAXJOBWALLTIME	
Format	`[[[DD:]HH:]MM:]SS`
Default	`-1` (NO LIMIT)
Description	The maximum amount of wallclock time that can be requested by any single job.
Example	`SYSTEMMAXJOBWALLTIME 1:00:00:00` *Moab will reject jobs requesting more than* `1 day` *of walltime.*

TARGETQUEUETIMEWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight assigned to the time remaining until the queuetime is reached.
Example	`TARGETQUEUETIMEWEIGHT 10`

TARGETWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The weight to be applied to a job's queuetime and expansion factor target components (see 4.1 Job Prioritization).
Example	`TARGETWEIGHT 1000`

TARGETXFACTORWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight assigned to the distance to the target expansion factor.
Example	`TARGETXFACTORWEIGHT   10`

TASKDISTRIBUTIONPOLICY	
Format	This parameter is deprecated and may be removed in a future release. (Regardless of how `TASKDISTRIBUTIONPOLICY` is configured, Moab always packs tasks by filling each node before moving to the next one in the list, in the order provided by the resource manager).

THREADPOOLSIZE	
Format	`<INTEGER>`
Default	`2X number of core processors` (MAX: 64)
Description	Governs the number of threads used when processing job scheduling. Scalability and performance might improve with multi-threading; to throttle, limit the number of threads used.
Example	`THREADPOOLSIZE 10`

TOOLSDIR	
Format	`<STRING>`
Default	`tools`
Description	The directory where Moab tools will be maintained (commonly used in conjunction with Native Resource Managers, and Triggers).
Example	`TOOLSDIR /var/adm/moab/tools`

TRACKSUSPENDEDJOBUSAGE	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Track the memory usage of suspended jobs on the nodes on which they are suspended and factor the memory usage into the scheduling of idle jobs.
Example	`TRACKSUSPENDEDJOBUSAGE TRUE` *Moab will track the memory usage of suspended jobs.*

TRAPFUNCTION	
Format	`<STRING>`
Default	---
Description	The functions to be trapped.
Example	`TRAPFUNCTION UpdateNodeUtilization|GetNodeSResTime`

TRAPJOB	
Format	`<STRING>`
Default	---
Description	The jobs to be trapped.
Example	`TRAPJOB pros23.0023.0`

TRAPNODE	
Format	<STRING>
Default	---
Description	The nodes to be trapped.
Example	`TRAPNODE node001|node004|node005`

TRAPRES	
Format	<STRING>
Default	---
Description	The reservations to be trapped.
Example	`TRAPRES interactive.0.1`

TRIGCHECKTIME	
Format	<INTEGER> (milliseconds)
Default	2000
Description	Each scheduling iteration, Moab will have a period of time where it handles commands and other UI requests. This time period is controlled by RMPOLLINTERVAL. During this time period, known as the UI phase, Moab will periodically evaluate triggers. Usually this only takes a fraction of a second, but if the number of triggers are large it could take up substantially more time (up to several seconds). While Moab is evaluating triggers, it doesn't respond to UI commands. This makes Moab feel sluggish and unresponsive. To remedy this, use the parameter TRIGCHECKTIME. This parameter tells Moab to only spend up to X milliseconds processing triggers during the UI phase. After X milliseconds has gone by, Moab will pause the evaluating of triggers, handle any pending UI events, and then restart the trigger evaluations where it last left off.
Example	`TRIGCHECKTIME 4000`

TRIGEVALLIMIT

Format	`<INTEGER>`
Default	`1`
Description	Each scheduling iteration, Moab will have a period of time where it handles commands and other UI requests. This time period is controlled by RMPOLLINTERVAL. During this time period, known as the UI phase, Moab will periodically evaluate triggers. The number of times Moab evaluates all triggers in the system is controlled by the `TRIGEVALLIMIT` parameter. By default, this is set to `1`. This means that Moab will evaluate all triggers at most once during the UI phase. Moab will not leave the UI phase and start other scheduling tasks until ALL triggers are evaluated at least one time. If TrigEvalLimit is set to `5`, then Moab will wait until all triggers are evaluated five times.
Example	`TRIGEVALLIMIT   3`

UIMANAGEMENTPOLICY

Format	One of `FORK` or `NONE`
Default	`NONE`
Description	When set with `FORK`, and with CLIENTUIPORT specified, Moab creates a new process to handle specific command requests in order to reduce command processing time. Currently, these commands are supported: • checkjob • showbf • showres • showstart See Appendix I: Considerations for Large Clusters for additional information on reducing command time (also known as low latency). This parameter should be configured on the server and any client machines.

Example	`UIMANAGEMENTPOLICY FORK`
	`CLIENTUIPORT 42560`

UJOBWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	Weight assigned by jobs per user. `-1` will reduce priority by number of active jobs owned by user.
Example	`UJOBWEIGHT 10`

UMASK	
Format	`<INTEGER>`
Default	`0022` (octal) (produces 0644 permissions)
Description	The file permission mask to use when creating new fairshare, stats, and event files. See the `umask` man page for more information.
Example	`UMASK 0127` *Create statistics and event files that are 'read-write' by owner and 'read' by group only.*

UNSUPPORTEDDEPENDENCIES	
Format	Comma-delimited string.
Default	---
Description	Specifies dependencies that are not supported and should not be accepted by job submissions. A maximum of 30 dependencies is supported.

Example	`# moab.cfg`
	 `UNSUPPORTEDDEPENDENCIES before,beforeok,beforenotok,on`
	
	 `> msub -l depend=before:105 cmd.sh`
	 `ERROR: cannot submit job - error in extension string`

UPROCWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	Weight assigned by processors per user. `-1` will reduce priority by number of active procs owned by user.
Example	`UPROCWEIGHT 10`

USAGECONSUMEDWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight assigned to per job processor second consumption.
Example	`USAGECONSUMEDWEIGHT 10`

USAGEEXECUTIONTIMEWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The priority weight assigned to the total job execution time (measured in seconds since job start). See Chapter 21: Preemption.
Example	`USAGEEXECUTIONTIMEWEIGHT 10`

USAGEPERCENTWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight assigned to total requested resources consumed.
Example	`USAGEPERCENTWEIGHT 5`

USAGEREMAININGWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight assigned to remaining usage.
Example	`USAGEREMAININGWEIGHT 10`

USAGEWEIGHT	
Format	`<INTEGER>`
Default	`1`
Description	The weight assigned to the percent and total job usage subfactors.
Example	`USAGEWEIGHT 100`

USEANYPARTITIONPRIO	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	The FSTREE data from the first feasible FSTREE will be used when determining a job's start priority, rather than having no FSTREE data considered. Do not set `USEANYPARTITIONPRIO` if you use per-partition scheduling. Doing so schedules jobs to the first partition listed, even if nodes from another partition will be available sooner.
Example	`USEANYPARTITIONPRIO TRUE`

USECPRSVNODELIST	
Format	`<BOOLEAN>`
Default	`TRUE`
Description	Specifies whether Moab should use the checkpointed reservation node list when rebuilding reservations on startup. If this is not used, then Moab will use the reservation's specified host expression during rebuilding.
Example	`USECPRSVNODELIST FALSE`

USEDATABASE	
Format	INTERNAL
Default	-
Description	Specifies whether Moab should store profile statistics, checkpoint information, and event information in an integrated database. See 2.2.2 Layout of Scheduler Components with Integrated Database Enabled for more information.
Example	`USEDATABASE INTERNAL`

USEJOBREGEX	
Format	BOOLEAN
Default	FALSE
Description	Specifies whether *mjobctl* supports regular expressions.

Example	
	```
	USEJOBREGEX TRUE

	[user@linux]$ mjobctl -c 8[1-3]

	job '81' canceled
	job '82' canceled
	job '83' canceled
	```

USEMOABCTIME	
Format	<BOOLEAN>
Default	FALSE
Description	When Moab finds new jobs on the resource manager, it creates a job inside of Moab for each job in the resource manager. By default, when Moab creates a new job, it uses the time the job was submitted to the resource manager to calculate how long the job has been in the queue (Moab processing time - job creation in resource manager), which is then used in determining the job's priority. In a system where more jobs are submitted to a resource manager than Moab can handle in one iteration, there is the possibility of jobs running out of order. For example, two jobs are both submitted at time 5. The first submitted job is processed first at time 6. So the first job's effective queue duration is 1 (6-5). On the next iteration, the second job is processed at time 8. So the second job's effective queue duration is 3 (8-5), indicating that it has been in the queue longer than the other job. Since the later job has a higher effective queue duration it will get a higher priority and could be scheduled to run before earlier submitted jobs. Setting `USEMOABCTIME` to `TRUE` tells Moab to use the creation time of the job in Moab rather than the creation time in the resource manager. This corrects the possible problem of having later submitted jobs having higher priorities and starting before earlier submitted jobs.
Example	`USEMOABCTIME TRUE`

USEMOABJOBID

Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether to return the Moab job ID when running 'msub', or the resource manager's job ID if it is available. `USEMOABJOBID` can also be set at the job level. The job level setting overrides this (global) setting in moabcfg. See the job flag USEMOABJOBID for more information.
Example	`USEMOABJOBID TRUE`

USERCFG[<USERID>]

Format	List of zero or more space delimited `<ATTR>=<VALUE>` pairs. Where `<ATTR>` is one of the following: General Credential Flags, `CDEF`, `DEFAULT.TPN`, `DEFAULT.WCLIMIT`, `EMAILADDRESS`, `ENABLEPROFILING`, `FSCAP`, `FSTARGET`, `JOBFLAGS`, `MAX.ARRAYSUBJOBS`, `MAX.WCLIMIT`, `NOEMAIL`, `OVERRUN`, `PLIST`, `PRIORITY`, `PRIVILEGES`, `QLIST`, `QDEF`, or a usage limit.
Default	---
Description	Specifies user specific attributes. For general user attribute information, See 2.7 Credentials. For a description of valid flag values, see 2.8 Job Flags.

Example	`USERCFG[john] MAXJOB=50 QDEF=highprio`
	`USERCFG[john] EMAILADDRESS=john@company.com`
	Up to `50` jobs submitted under the user ID `john` will be allowed to execute simultaneously and will be assigned the QOS `highprio`.

USERPRIOCAP	
Format	`<INTEGER>`
Default	---
Description	The priority cap to be applied to the user specified job priority factor. Under Moab, only negative user priorities can be specified. See 4.1.2.E Service (SERVICE) Component.

Example	`USERPRIOWEIGHT 10`
	`USERPRIOCAP    -10000`

USERPRIOWEIGHT	
Format	`<INTEGER>`
Default	1
Description	The weight to be applied to the user specified job priority. Under Moab, only negative user priorities can be specified. If this weight is set, users can reduce the priority of some of their jobs to allow other jobs to run earlier. See 4.1.2.E Service (SERVICE) Component and 4.1.4.D User Selectable Prioritization. If Viewpoint is part of your configuration, this value must be at least 1. Otherwise, Moab will not take into consideration any user priority information specified for a job that was created using Viewpoint.
Example	`USERPRIOWEIGHT 10`

USERWEIGHT	
Format	`<INTEGER>`
Default	1
Description	The weight to be applied to the user priority of each job. See 4.1.2.B Credential (CRED) Component.
Example	`USERWEIGHT 10`

USESYSLOG	
Format	`<BOOLEAN>[<FACILITY>]`
Default	`FALSE:daemon`
Description	Specifies whether or not the scheduler will report key events to the system syslog facility. If the `<FACILITY>` is specified, Moab will report events to this syslog facility. See 12.2 Logging for more information.
Example	`USESYSLOG TRUE:local3` *Moab will report key events, commands, and failures to syslog using the `local3` facility.*

USESYSTEMQUEUETIME	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Specifies whether or not job prioritization should be based on the time the job has been eligible to run (i.e., idle and meets all fairness policies [`TRUE`] or the time the job has been idle [`FALSE`]). See 4.1.2 Job Priority Factors for more information. **Note**: This parameter has been superseded by the JOBPRIOACCRUALPOLICY parameter.
Example	`USESYSTEMQUEUETIME FALSE` *The queuetime and expansion factor components of a job's priority will be calculated based on the length of time the job has been in the idle state.*

USEUSERHASH	
Format	`<BOOLEAN>`
Default	`FALSE`
Description	Enables searching of the user buffer using the user hash key instead of doing sequential searches of the user buffer.
Example	`USEUSERHASH TRUE`

WALLTIMECAP

Format	`<DOUBLE>`
Default	`0` (NO CAP)
Description	The maximum total preweighted absolute contribution to job priority that can be contributed by the walltime component. This value is specified as an absolute priority value, not as a percent.
Example	`WALLTIMECAP 10000` *Moab will bound a job's preweighted walltime priority component within the range +/- `10000`.*

WALLTIMEWEIGHT

Format	`<INTEGER>`
Default	`0`
Description	The priority weight to be applied to the amount of walltime requested by a job (in seconds). See 4.1.2.D Resource (RES) Component.

Example	`RESWEIGHT      10`
	 `WALLTIMEWEIGHT 100`
	 Increase the priority of longer duration jobs.

WCACCURACYCAP

Format	`<DOUBLE>`
Default	`0` (NO CAP)
Description	The maximum total preweighted absolute contribution to job priority that can be contributed by the wallclock accuracy component. This value is specified as an absolute priority value, not as a percent.
Example	`WCACCURACYCAP 10000` *Moab will bound a job's preweighted wallclock accuracy priority component within the range +/- `10000`.*

WCACCURACYWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The priority weight to be applied to the job's historical user wallclock accuracy (range 0.0 to 1.0). See 4.1.2.C Fairshare (FS) Component.

Example	`FSWEIGHT         10`
	`WCACCURACYWEIGHT 100`
	Favor jobs with good wallclock accuracies by giving them a priority increase.

WCVIOLATIONACTION	
Format	One of `CANCEL` or `PREEMPT`.
Default	`CANCEL`
Description	The action to take when a job exceeds its wallclock limit. If set to `CANCEL`, the job will be terminated. If set to `PREEMPT`, the action defined by the PREEMPTPOLICY parameter will be taken. See the parameter JOBMAXOVERRUN or 5.2.6 Usage-Based Limits.

Example	`WCVIOLATIONACTION   PREEMPT`
	`PREEMPTPOLICY       REQUEUE`
	Moab will requeue jobs that exceed their wallclock limit.

WEBSERVICESURL	
Format	`<URL>`
Default	---
Description	If specified, Moab sends data to Moab Web Services (MWS) to be stored in a database. This allows Moab to spend more cycles on scheduling instead of database interaction. The sending occurs via HTTP PUT.
Example	`WEBSERVICESURL https://mws-staging.ac:8080/mws/rm/moab/dump` *Moab sends data that needs to be stored in a database to the specified URL.*

WIKIEVENTS	
Format	`<BOOLEAN>` This parameter is deprecated and may be removed in a future release.

XFACTORCAP	
Format	`<DOUBLE>`
Default	`0` (NO CAP)
Description	The maximum total preweighted absolute contribution to job priority that can be contributed by the expansion factor component. This value is specified as an absolute priority value, not as a percent.
Example	`XFACTORCAP 10000` *Moab will bound a job's preweighted XFactor priority component within the range +/- `10000`.*

XFACTORWEIGHT	
Format	`<INTEGER>`
Default	`0`
Description	The weight to be applied to a job's minimum expansion factor before it is added to the job's cumulative priority.
Example	`XFACTORWEIGHT 1000` *Moab will multiply a job's XFactor value by* `1000` *and then add this value to its total priority.*

XFMINWCLIMIT	
Format	`[[[DD:]HH:]MM:]SS`
Default	`-1` (NO LIMIT)
Description	The minimum job wallclock limit that will be considered in job expansion factor priority calculations.
Example	`XFMINWCLIMIT 0:01:00` *Jobs requesting less than 1 minute of wallclock time will be treated as if their wallclock limit was set to* `1 minute` *when determining expansion factor for priority calculations.*

Appendix B: Multi-OS Provisioning

In this appendix:

Moab can dynamically provision compute machines to requested operating systems and power off compute machines when not in use. Moab can intelligently control xCAT and use its advanced system configuration mechanisms to adapt systems to current workload requirements. Moab communicates with xCAT using the Moab Service Manager (MSM). MSM is a translation utility that resides between Moab and xCAT and acts as aggregator and interpreter. Moab Workload Manager will query MSM, which in turn queries xCAT, about system resources, configurations, images, and metrics. After learning about these resources from MSM, Moab then makes intelligent decisions about the best way to maximize system utilization.

In this model, Moab gathers system information from two resource managers. The first is Torque, which handles the workload on the system; the second is MSM, which relays information gathered by xCAT. By leveraging these software packages, Moab intelligently adapts clusters to deliver on-site goals.

This document assumes that xCAT has been installed and configured. It describes the process of getting MSM and xCAT communicating, and it offers troubleshooting guidance for basic integration. This document offers a description for how to get Moab communicating with MSM and the final steps in verifying a complete software stack.

B.1 xCAT Plug-in Configuration Parameters

Plug-in parameters that begin with an underscore character are specific to the xCAT plug-in; others are common to all plug-ins and can either be set in the `RMCFG[msm]` for all plug-ins, or per plug-in in the `APPCFG[<plugin_name>]`.

_CQxCATSessions _DoNodeStat _DORVitals _DoxCATStats _ESXStore _FeatureGroups _HVxCATPasswdKey _ImagesTabName _LockDir	_MaskOSWhenOff _ModifyTorque _NodeRange _NoRollbackOnError _PowerString _QueueRPower _ReportNETADDR _RPowerQueueAge _RPowerQueueSize	_RPowerTimeOut _UseStates _VerifyRPower _xCATHost Description LogLevel Module PollInterval TimeOut

_CQxCATSessions	
Format	Positive integer > 1
Default	`10`
Description	MSM will divide the node list generated by *`nodels`* into this many groups and simultaneously query xCAT for each group. The value may need tuning for large installations, higher values will cause the time to complete a single cluster query to go down, but cause a higher load on the xCAT headnode.

_DoNodeStat	
Format	`0` or `1`
Default	`1`
Description	If set to `0`, MSM will not call *`nodestat`* to generated a substate. This can be used to speed up the time it takes to query xCAT, and you do not need the substate visible to Moab.

_DORVitals	
Format	`0` or `1`
Default	`0`

_DORVitals	
Description	When set to 1, MSM will poll rvitals power and led status (see the xCAT rvitals man page). This only works with IBM BMCs currently. In order to use this, xCAT should respond without error to the *rvitals <noderange> watts* and *rvitals <noderange> leds* commands. Status is reported as GMETRTIC[watts] and GMETRIC[leds]. See also the _PowerString configuration parameter.

_DoxCATStats	
Format	0 or 1
Default	0
Description	If Set to 1, MSM will track performance statistics about calls to xCAT, and the performance of higher level operations. The information is available via the script $MSMHOMEDIR/contrib/xcat/xcatstats.pl. This parameter is useful for tuning the POLLINTERVAL and _CQxCATSessions configuration parameters.

_ESXStore	
Format	Mountable NFS Path
Default	---
Description	Location of ESX stores.

_FeatureGroups	
Format	Comma-delimited string of xCAT group names.
Default	---
Description	MSM builds the OSLIST for a node as the intersection of _FEATUREGROUPS, features specified in x_msm for that image, and the nodes group membership. The value 'torque' is special, and indicates that the image uses Torque, and the node should be added/removed from Torque during provisioning when used in conjunction with the _ModifyTorque parameter.

_HVxCATPasswdKey	
Format	key value in the xCAT passwd table
Default	`vmware`
Description	This is where MSM gets the user/password to communicate with ESX hypervisors.

_ImagesTabName	
Format	Existing xCAT table that contains your image definitions.
Default	`x_msm`
Description	This table specifies the images that can be presented to Moab in a node's `OSLIST`. The xCAT schema for this table is defined in `$MSMHOMEDIR/contrib/xcat/MSM.pm`, which needs to be copied to the `$XCATROOT/lib/perl/xCAT_schema` directory.

_LockDir	
Format	Existing path on MSM host.
Default	`$MSMHOMEDIR/lck`
Description	This is a path to where MSM maintains lock files to control concurrency with some Xen and KVM operations.

_MaskOSWhenOff	
Format	`0` or `1`
Default	`0`
Description	When set, this parameter will cause MSM to report `OS=None` for nodes that are powered off. This may be useful when mixing stateless and stateful images, forcing Moab to request provisioning instead of just powering on a node.

_ModifyTorque	
Format	`0` or `1`
Default	`0`
Description	When set, this parameter will cause MSM to add and removes nodes from Torque as required by provisioning. See the _FeatureGroups parameter also.

_NodeRange	
Format	Any valid noderange (see the xCAT noderange man page).
Default	All
Description	When MSM queries xCAT this is the noderange it will use. At sites where xCAT manages other hardware that Moab is not intended to control, it is important to change this.

_NoRollbackOnError	
Format	`0` or `1`
Default	`0`
Description	When an error occurs and rollback is activated (as it is by default), rollback causes a reversion to the previous successful request. `_NoRollbackOnError` is useful for debugging to determine the xCAT state if no rollback occurred. If set to 1 and an error occurs between MSM and xCAT when creating a node, assigning a name (DNS) to a node, or assigning an IP address (DHCP) to a node, then no rollback occurs.

_PowerString	
Format	single quote delimited string
Default	`'AC Avg Power'`
Description	Only meaningful when used with _DORVitals=`1`. Some BMCs return multiple responses to the rvitals command, or use slightly different text to describe the power metrics. Use this parameter to control what is reported to Moab. You can use `'$MSMLIBDIR/contrib/xcat/dump.xcat.cmd.pl`*`rvitals`*

_PowerString	
	`<node_name> power'` and examine the output to determine what the appropriate value of this string is.

_QueueRPower	
Format	`0` or `1`
Default	`0`
Description	When set, this parameter will cause MSM to aggregate rpower requests to xCAT into batches. The timing and size of these batches is controlled with the _RPowerQueueAge and _RPowerQueueSize parameters. **Note**: This can significantly reduce load on the xCAT headnode, but will cause the power commands to take longer, and MSM shutdown to take longer.

_ReportNETADDR	
Format	`0` or `1`
Default	`0`
Description	When set, this parameter will cause MSM to report NETADDR=<hosts.ip from xCAT>.

_RPowerQueueAge	
Format	Positive integer values
Default	`30`
Description	Only meaningful when used with _QueueRPower. MSM will send any pending rpower requests when the oldest request in the queue exceeds this value (seconds).

_RPowerQueueSize	
Format	Positive integer values

_RPowerQueueSize	
Default	`200`
Description	Only meaningful when used with _QueueRPower. MSM will send any pending rpower requests when the queue depth exceeds this value.

_RPowerTimeOut	
Format	Positive integer values
Default	`60`
Description	Only meaningful when used with _VerifyRPower. If nodes do not report the expected power state in this amount of time, a GEVENT will be produced on the node (or system job).

_UseStates	
Format	Valid xCAT chain.currstate values (see the xCAT chain man page)
Default	`boot,netboot,install`
Description	Nodes that do not have one of these values in the xCAT `chain.currstate` field will reported with `STATE=Updating`. Use this configuration parameter to prevent Moab from scheduling nodes that are updating firmware, etc.

_VerifyRPower	
Format	`0` or `1`
Default	`0`
Description	If set, MSM will attempt to confirm that rpower requests were successful by polling the power state with rpower stat until the node reports the expected state, or _RPowerTimeOut is reached. **Note**: This can create significant load on the xCAT headnode.

_xCATHost	
Format	`<xcat_headnode>:<xcatd_port>`

_xCATHost	
Default	`localhost:3001`
Description	Use to configure MSM to communicate with xCAT on another host.

Description	
Format	Double quoted string containing brief description of plug-in.
Default	---
Description	This information is not visible in Moab, but shows up in *`msmctl -a`*.

LogLevel	
Format	1-9
Default	`5`
Description	Used to control the verbosity of logging, 1 being the lowest (least information logged) and 9 being the highest (most information logged). For initial setup and testing, 8 is recommended, then lowering to 3 (only errors logged) for normal operation. Use 9 for debugging, or when submitting a log file for support.

Module	
Format	Moab::MSM::App::xCAT
Default	---
Description	Name of the plug-in module to load.

PollInterval	
Format	Integer > 0
Default	`60`
Description	MSM will query xCAT every `POLLINTERVAL` seconds to update general node status. This number will likely require tuning for each specific system. In

PollInterval	
	general, to develop this number, you should pick a fraction of the total nodes MSM will be managing (1/_CQXCATSESSIONS), and time how long it takes run nodestat, rpower stat, and optionally rvitals on these nodes, and add ~15%. Increasing the `POLLINTERVAL` will lower the overall load on the xCAT headnode, but decrease the responsiveness to provisioning and power operations.

TimeOut	
Format	Integer value > POLLINTERVAL
Default	`300`
Description	This parameter controls how long MSM will wait for child processed to complete (all xCAT commands are run in child processes). After `TIMEOUT` seconds, if a child has not returned it will be killed, and an error reported for the operation.

B.2 Configuration Validation

Set up environment to manually call MSM commands:

```
# substitute appropriate value(s) for path(s)
  export MSMHOMEDIR=/opt/moab/tools/msm
  export MSMLIBDIR=/opt/moab/tools/msm
  export PATH=$PATH:/$MSMLIBDIR/contrib:$MSMLIBDIR/bin
```

Verify that MSM starts without errors:

```
> msmd
```

Verify that the expected nodes are listed, without errors, using the value of `_NODERANGE` from `msm.cfg`:

```
> nodels <_NODERANGE>
```

Verify that the expected nodes, are listed in the cluster query output from MSM:

```
> cluster.query.pl
```

Provision all nodes through MSM for the first time (pick and image name from `x_msm`):

```
> for i in `nodels <_NODERANGE>; do node.modify.pl $i --set os=<image_name>;done
```

Verify the nodes correctly provision and that the correct OS is reported (which might take some time after the provisioning requests are made):

```
> cluster.query.pl
```

B.3 Deploying Images with Torque

When using MSM + xCAT to deploy images with Torque, there are some special configuration considerations. Most of these also apply to other workload resource managers.

Note that while the MSM xCAT plug-in contains support for manipulating Torque directly, this is not an ideal solution. If you are using a version of xCAT that supports prescripts, it is more appropriate to write prescripts that manipulate Torque based on the state of the xCAT tables. This approach is also applicable to other workload resource managers, while the xCAT plug-in only deals with Torque.

Several use cases and configuration choices are discussed in what follows.

Each image should be configured to report its image name through Torque. In the Torque *`pbs_mom`* `mom_config` file the `opsys` value should mirror the name of the image. See 'Appendix C: Node Manager (MOM) Configuration' in the *Torque Resource Manager Administrator Guide* for more information.

B.4 Installing Moab on the Management Node

Moab is the intelligence engine that coordinates the capabilities of xCAT and Torque to dynamically provision compute nodes to the requested operating system. Moab also schedules workload on the system and powers off idle nodes. Download and install Moab.

B.5 Integrating MSM and xCAT

Copy the `x_msm` table schema to the xCAT schema directory:

```
> cp $MSMHOMEDIR/contrib/xcat/MSM.pm $XCATROOT/lib/perl/xCAT_schema
```

Restart *`xcatd`* and check the `x_msm` table is correctly created:

```
> service xcatd restart
```

```
> tabdump x_msm
```

Prepare xCAT images and ensure they provision correctly (see xCAT documentation).

Populate the `x_msm` table with your image definitions:

```
> tabedit x_msm

#flavorname,arch,profile,os,nodeset,features,vmoslist,hvtype,hvgroupname,vmgroupname,c
omments,disable
  "compute","x86_64","compute","centosX.X","netboot","torque",,,,,,
  "science","x86","compute","scientific_linux","netboot","torque",,,,,,
```

- **flavorname** - A user specified name for the image and settings; also an xCAT group name, nodes are added to this group when provisioned
- **arch** - Architecture as used by xCAT
- **profile** - Profile as used by xCAT
- **os** - Operating system as used by xCAT
- **nodeset** - One of netboot|install|statelite
- **features** - Names of xCAT groups that identify special hardware features ('torque' and 'paravirt' are special cases)
- **vmoslist** - Note: Not used. List of flavornames this image can host as VMs (hypervisor images only)
- **hvtype** - Note: Not used. One of esx|xen|kvm (hypervisor images only)
- **hvgroupname** - Note: Not used. Name of xCAT group nodes will be added to when provisioned to this image
- **vmgroupname** - Note: Not used. Name of xCAT group VMs will be added to when hosted on a hypervisor of this image
- **comments** - User specified comments
- **disable** - Flag to temporarily disable use of this image

Ensure all xCAT group names in the x_msm table exist in the xCAT nodegroup table:

```
> tabedit nodegroup
```

Edit as necessary to simulate the following example:

```
  #groupname,grouptype,members,wherevals,comments,disable
  "compute",,,,,
  "esxi4",,,,,
  "esxhv",,,,,
  "esxvmmgt",,,,,
```

After making any necessary edits, run the following command:

```
> nodels compute,esxi4,esxhv,esxvmmgt
  # should complete without error, ok if doesn't return anything
```

B.6 Moab Configuration File Example

Moab stores its configuration in the `moab.cfg` file: `/opt/moab/etc/moab.cfg`. A sample configuration file, set up and optimized for Adaptive Computing follows:

```
SCHEDCFG[Moab]           SERVER=gpc-sched:42559
ADMINCFG[1]              USERS=root,egan
LOGLEVEL                 7

# How often (in seconds) to refresh information from Torque and MSM
RMPOLLINTERVAL            60,60
RESERVATIONDEPTH         10
DEFERTIME                0
TOOLSDIR                 /opt/moab/tools

###############################################################################
# Torque and MSM configuration                                                #
###############################################################################
RMCFG[torque]            TYPE=PBS
RMCFG[msm]         TYPE=NATIVE:msm FLAGS=autosync,NOCREATERESOURCE RESOURCETYPE=PROV
RMCFG[msm]         TIMEOUT=60
RMCFG[msm]         PROVDURATION=10:00
AGGREGATENODEACTIONS     TRUE

###############################################################################
# ON DEMAND PROVISIONING SETUP                                                #
###############################################################################
QOSCFG[od]               QFLAGS=PROVISION
USERCFG[DEFAULT]         QLIST=od
NODEALLOCATIONPOLICY     PRIORITY
NODECFG[DEFAULT]         PRIORITYF=1000*OS+1000*POWER
NODEAVAILABILITYPOLICY   DEDICATED
CLASSCFG[DEFAULT]        DEFAULT.OS=scinetcompute

###############################################################
# GREEN POLICIES                                              #
###############################################################
NODECFG[DEFAULT]         POWERPOLICY=ONDEMAND
PARCFG[ALL]              NODEPOWEROFFDURATION=20:00
NODEIDLEPOWERTHRESHOLD   600
# END Example moab.cfg
```

B.7 MSM Configuration

Edit `$MSMHOMEDIR/msm.cfg` and configure the xCAT plug-in. Below is a generic example for use with Torque without virtualization. See B.1 xCAT Plug-in Configuration Parameters for a complete list of parameters and descriptions.

```
  # MSM configuration options
  RMCFG[msm]         PORT=24603
  RMCFG[msm]         POLLINTERVAL=45
```

```
RMCFG[msm]          LOGFILE=/opt/moab/log/msm.log
RMCFG[msm]          LOGLEVEL=8
RMCFG[msm]          DEFAULTNODEAPP=xcat

# xCAT plugin specific options
APPCFG[xcat]        DESCRIPTION="xCAT plugin"
APPCFG[xcat]        MODULE=Moab::MSM::App::xCAT
APPCFG[xcat]        LOGLEVEL=3
APPCFG[xcat]        POLLINTERVAL=45
APPCFG[xcat]        TIMEOUT=3600
APPCFG[xcat]        _USEOPIDS=0
APPCFG[xcat]        _NODERANGE=moab,esxcompute
APPCFG[xcat]        _USESTATES=boot,netboot,install
APPCFG[xcat]        _LIMITCLUSTERQUERY=1
APPCFG[xcat]        _RPOWERTIMEOUT=120
APPCFG[xcat]        _DONODESTAT=1
APPCFG[xcat]        _REPORTNETADDR=1
APPCFG[xcat]        _CQXCATSESSIONS=4
```

B.8 MSM Installation

- Determine the installation directory (usually `/opt/moab/tools/msm`)
- Untar the MSM tarball into the specified directory (making it the MSM home directory, or `$MSMHOMEDIR`)
- Verify the required Perl modules and version are available:

```
> perl -e 'use Storable 2.18'
> perl -MXML::Simple -e 'exit'
> perl -MProc::Daemon -e 'exit'
> perl -MDBD::SQLite -e 'exit'
```

B.9 Troubleshooting

- **msmctl -a does not report the xCAT plug-in** - Check the log file (path specified in `msm.cfg`) for error messages. A common cause is missing Perl modules (Storable, DBD::SQLite, xCAT::Client).
- **cluster.query.pl does not report any nodes** - Check that the xCAT command *`nodels`*`<noderange>`, where `<noderange>` is the value configured for _ NODERANGE in msm.cfg, outputs the nodes expected.
- **cluster.query.pl does not report OS** - MSM must provision a node to recognize what the current operating system is. It is not sufficient to look up the values in the `nodetype` table because MSM has no way of recognizing whether *`nodeset`* and *`rpower`* were run with the current values in the `nodetype` table.

- **cluster.query.pl does not report OSLIST, or does not report the expected OSLIST for a node** - Check that the node belongs to the appropriate groups, particularly any listed in the features field of the `x_msm` table for the missing image name.

B.10 Verifying the Installation

When Moab starts, it immediately communicates with its configured resource managers. In this case Moab communicates with Torque to get compute node and job queue information. It then communicates with MSM to determine the state of the nodes according to xCAT. It aggregates this information and processes the jobs discovered from Torque.

When a job is submitted, Moab determines whether nodes need to be provisioned to a particular operating system to satisfy the requirements of the job. If any nodes need to be provisioned Moab performs this action by creating a provisioning system job (a job that is internal to Moab). This system job communicates with xCAT to provision the nodes and remain active while the nodes are provisioning. Once the system job has provisioned the nodes it informs the user's job that the nodes are ready at which time the user's job starts running on the newly provisioned nodes.

When a node has been idle for a specified amount of time (see the parameter NODEIDLEPOWERTHRESHOLD), Moab creates a power-off system job. This job communicates with xCAT to power off the nodes and remains active in the job queue until the nodes have powered off. Then the system job informs Moab that the nodes are powered off but are still available to run jobs. The power off system job then exits.

To verify correct communication between Moab and MSM run the *`mdiag -R -v`* msm command:

```
$ mdiag -R -v msm
diagnosing resource managers
RM[msm]       State: Active  Type: NATIVE:MSM  ResourceType: PROV
  Timeout:            30000.00 ms
  Cluster Query URL:  $HOME/tools/msm/contrib/cluster.query.xcat.pl
  Workload Query URL: exec://$TOOLSDIR/msm/contrib/workload.query.pl
  Job Start URL:      exec://$TOOLSDIR/msm/contrib/job.start.pl
  Job Cancel URL:     exec://$TOOLSDIR/msm/contrib/job.modify.pl
  Job Migrate URL:    exec://$TOOLSDIR/msm/contrib/job.migrate.pl
  Job Submit URL:     exec://$TOOLSDIR/msm/contrib/job.submit.pl
  Node Modify URL:    exec://$TOOLSDIR/msm/contrib/node.modify.pl
  Node Power URL:     exec://$TOOLSDIR/msm/contrib/node.power.pl
  RM Start URL:       exec://$TOOLSDIR/msm/bin/msmd
  RM Stop URL:        exec://$TOOLSDIR/msm/bin/msmctl?-k
  System Modify URL:  exec://$TOOLSDIR/msm/contrib/node.modify.pl
  Environment:
MSMHOMEDIR=/home/wightman/test/scinet/tools//msm;MSMLIBDIR=/home/wightman/test/scinet/
tools//msm
  Objects Reported:   Nodes=10 (0 procs)  Jobs=0
  Flags:              autosync
  Partition:          SHARED
  Event Management:   (event interface disabled)
  RM Performance:     AvgTime=0.10s  MaxTime=0.25s  (38 samples)
  RM Languages:       NATIVE
  RM Sub-Languages:   -
```

To verify nodes are configured to provision use the *`checknode -v`* command (each node will have a list of available operating systems):

```
$ checknode n01
node n01
State:      Idle  (in current state for 00:00:00)
Configured Resources: PROCS: 4  MEM: 1024G  SWAP: 4096M  DISK: 1024G
Utilized   Resources: ---
Dedicated  Resources: ---
Generic Metrics:    watts=25.00,temp=40.00
Power Policy:       Green (global policy)   Selected Power State: Off
Power State:   Off
Power:      Off
  MTBF(longterm):   INFINITY  MTBF(24h):   INFINITY
Opsys:      compute   Arch:      ---
  OS Option: compute
  OS Option: computea
  OS Option: gpfscompute
  OS Option: gpfscomputea
Speed:      1.00      CPULoad:   0.000
Flags:      rmdetected
RM[msm]:    TYPE=NATIVE:MSM  ATTRO=POWER
EffNodeAccessPolicy: SINGLEJOB
Total Time: 00:02:30  Up: 00:02:19 (92.67%)  Active: 00:00:11 (7.33%)
```

To verify nodes are configured for Green power management, run the *`mdiag -G`* command (each node will show its power state):

```
$ mdiag -G
NOTE:  power management enabled for all nodes
Partition ALL:  power management enabled
  Partition NodeList:
Partition local:  power management enabled
  Partition NodeList:
  node n01 is in state Idle, power state On (green powerpolicy enabled)
  node n02 is in state Idle, power state On (green powerpolicy enabled)
  node n03 is in state Idle, power state On (green powerpolicy enabled)
  node n04 is in state Idle, power state On (green powerpolicy enabled)
  node n05 is in state Idle, power state On (green powerpolicy enabled)
  node n06 is in state Idle, power state On (green powerpolicy enabled)
  node n07 is in state Idle, power state On (green powerpolicy enabled)
  node n08 is in state Idle, power state On (green powerpolicy enabled)
  node n09 is in state Idle, power state On (green powerpolicy enabled)
  node n10 is in state Idle, power state On (green powerpolicy enabled)
Partition SHARED:  power management enabled
```

To submit a job that dynamically provisions compute nodes, run the *`msub -l os=<image>`* command:

```
$ msub -l os=computea job.sh
yuby.3
$ showq
active jobs------------------------
```

```
JOBID                USERNAME      STATE PROCS   REMAINING            STARTTIME
provision-4              root    Running     8    00:01:00  Fri Jun 19 09:12:56
1 active job                 8 of 40 processors in use by local jobs (20.00%)
                             2 of 10 nodes active      (20.00%)
eligible jobs----------------------
JOBID                USERNAME      STATE PROCS    WCLIMIT            QUEUETIME
yuby.3               wightman       Idle     8    00:10:00  Fri Jun 19 09:12:55
1 eligible job
blocked jobs-----------------------
JOBID                USERNAME      STATE PROCS    WCLIMIT            QUEUETIME

0 blocked jobs
Total jobs:  2
```

Notice that Moab created a provisioning system job named `provision-4` to provision the nodes. When `provision-4` detects that the nodes are correctly provisioned to the requested OS, the submitted job `yuby.3` runs:

```
$ showq
active jobs------------------------
JOBID                USERNAME      STATE PROCS   REMAINING            STARTTIME
yuby.3               wightman    Running     8    00:08:49  Fri Jun 19 09:13:29
1 active job                 8 of 40 processors in use by local jobs (20.00%)
                             2 of 10 nodes active      (20.00%)
eligible jobs----------------------
JOBID                USERNAME      STATE PROCS    WCLIMIT            QUEUETIME

0 eligible jobs
blocked jobs-----------------------
JOBID                USERNAME      STATE PROCS    WCLIMIT            QUEUETIME

0 blocked jobs
Total job:  1
```

The *`checkjob`* command shows information about the provisioning job and the submitted job. If any errors occur, run the *`checkjob -v <jobid>`* command to diagnose failures.

B.11 xCAT Configuration Requirements

Observe the following xCAT configuration requirements before installing MSM:

- Configure xCAT normally for your site.
    - Test the following commands to verify proper function:
        - rpower
        - nodeset
        - makedhcp
        - makedns

        - nodestat
        - rvitals
    - If MSM will run on a different machine than the one on which xCAT runs, install the xCAT client packages on that machine, and test the previously listed commands on that machine also.
    - Configure and test all stateful/stateless images you intend to use.
- Configure xCAT to use either PostgreSQL or MySQL. Note that the default of SQLite might not function properly when MSM drives xCAT.
    - PostgreSQL: See the web page Setting Up PostgreSQL as the xCAT DB for more information.
    - MySQL: See the web page Setting Up MySQL as the xCAT DB for more information.

You must have a valid Moab license file (`moab.lic`) with provisioning and green enabled. For information on acquiring an evaluation license, contact info@adaptivecomputing.com.

Appendix C: Moab Event Dictionary

See 12.2 Logging for more information about Moab logging.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1000005	USER	system.moab	INFO	MWM_TESTING_INFO	Testing with argument1: %s. and argument2: %s and argument3: %s and argument4: %s	Internal error for testing diagnostics.
0x1000065	USER	domain.lifecycle	INFO	MWM_JOB_CANCEL	Job %s was canceled. %s	The job was canceled.
0x1000066	USER	domain.lifecycle	INFO	MWM_JOB_END_SUCCESSFUL	Job %s finished successfully at %s.	The job finished successfully.
0x1000068	USER	domain.lifecycle	INFO	MWM_JOB_USER_HOLD	Job %s had a user hold applied.	A user hold was applied to the job.
0x1000069	USER	domain.lifecycle	INFO	MWM_JOB_SYSTEM_HOLD	Job %s had a system hold applied.	A system hold was applied to the job.
0x100006a	USER	domain.lifecycle	INFO	MWM_JOB_BATCH_HOLD	Job %s had a batch hold applied.	A batch hold was applied to the job.
0x100006b	USER	domain.lifecycle	INFO	MWM_JOB_DEFER_HOLD	Job %s had a defer hold applied.	A defer hold was applied to the job.
0x100006c	USER	domain.lifecycle	INFO	MWM_JOB_MODIFY	Job %s was modified. %s	One of the attributes of the job was modified either

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						via a user initiated action or an automated action.
0x1000006d	USER	domain.lifecycle	INFO	MWM_JOB_REJECT	Job %s was rejected. %s	The job was rejected for some reason.
0x1000006e	USER	domain.lifecycle	INFO	MWM_JOB_RELEASE	Job %s was released.	Any holds placed on the job have been released, and the job is not prevented from running due to any hold action. The job might still not be able to run due to other considerations.
0x1000006f	USER	domain.lifecycle	INFO	MWM_JOB_START	Job %s was started. %s	The job was started on its designated node[s].
0x10000070	USER	domain.lifecycle	INFO	MWM_JOB_SUBMIT	Job %s was submitted. %s	The job has been submitted to Moab and is being evaluated and processed.
0x10000071	USER	domain.lifecycle	INFO	MWM_JOB_CREATED	Job %s was created.	The job has been created and will be queued for execution.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1000072	USER	domain.lifecycle	INFO	MWM_JOB_REQUEUE	Job %s was requeued. %s	The job has been requeued so it can be executed again.
0x1000073	USER	domain.lifecycle	INFO	MWM_JOB_CANCEL_CLEANUP_STARTED	Job %s is being cleaned up due to cancel request.	The job has been issued a cancel request and is being cleaned up.
0x1000074	USER	domain.lifecycle	INFO	MWM_JOB_CLEANUP_STARTED	Job %s is being cleaned up.	The job has ended and is being cleaned up.
0x1000075	USER	domain.lifecycle	INFO	MWM_JOB_DEFERRED	Job %s has been deferred.	The job has been deferred.
0x1000076	USER	domain.lifecycle	INFO	MWM_JOB_RENAME	Job %s has been renamed to %s.	The job has been renamed.
0x10000c9	USER	domain.lifecycle	INFO	MWM_NODE_EVAC_VMS	Evacuating VMs off node %s.	Evacuating VMs off the node.
0x100012c	USER	domain.lifecycle	INFO	MWM_RSV_CREATE	Reservation %s was created. %s	The reservation has been created and is stored in the system.
0x100012d	USER	domain.lifecycle	INFO	MWM_RSV_START	Reservation %s has started.	The reservation has started.
0x1000	USER	domain.lifec	INF	MWM_RSV_END	Reservation %s has ended.	The reservation has

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
12e		ycle	O			ended.
0x1000190	USER	system.moab	INFO	MWM_SCHED_COMMAND	The following scheduler command was submitted: %s	External commands are submitted to Moab in a variety of ways. This event documents the command line and possibly other information associated with the command. These commands typically have the ability to change behavior/state within Moab. Commands that are typically queries are not included.
0x1000192	USER	system.moab	INFO	MWM_SCHED_CYCLE_START	A scheduler iteration is beginning. %s	Moab periodically checks through submitted jobs and makes decisions regarding which jobs are scheduled. One of these iterations is beginning now.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1000193	USER	system.moab	INFO	MWM_SCHED_CYCLE_END	A scheduler iteration is ending. %s	Moab periodically checks through submitted jobs and makes decisions regarding which jobs are scheduled. One of these iterations is ending now.
0x1000194	USER	system.moab	INFO	MWM_SCHED_PAUSE	The scheduler has been paused. %s	The Moab scheduler has been administratively paused. New jobs can be submitted and existing jobs will continue to run, but no new jobs will be scheduled as long as Moab is paused.
0x1000195	USER	system.moab	INFO	MWM_SCHED_RECYCLE	The scheduler has been recycled. %s	The Moab scheduler has been administratively recycled. The process will cleanly exit and save the state data. It will then restart, read in the data, and resume

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						scheduling.
0x1000196	USER	system.moab	INFO	MWM_SCHED_RESUME	The scheduler has been resumed.	The Moab scheduler has been administratively resumed. A new scheduling iteration will begin immediately and continue regularly.
0x1000197	USER	system.moab	INFO	MWM_SCHED_START	The scheduler has started.	The Moab scheduler has started.
0x1000198	USER	system.moab	INFO	MWM_SCHED_STOP	The scheduler has stopped. %s	The Moab scheduler has stopped.
0x10001f4	USER	domain.lifecycle	INFO	MWM_TRIG_CREATE	Trigger %s has been created.	The named trigger has been created and is now recognized in the Moab system.
0x10001f5	USER	domain.lifecycle	INFO	MWM_TRIG_START	Trigger %s has started.	The named trigger has started its action.
0x10001f6	USER	domain.lifecycle	INFO	MWM_TRIG_END	Trigger %s has ended. %s	The named trigger has finished its action.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x10001f8	USER	domain.lifecycle	INFO	MWM_TRIG_THRESHOLD	Trigger %s threshold event: %s	A trigger threshold has been encountered. Additional details regarding the threshold might be included in the text.
0x1000258	USER	domain.lifecycle	INFO	MWM_VM_SUBMIT	VM %s has been submitted.	The named VM has been submitted and is now recognized in the Moab system.
0x1000259	USER	domain.lifecycle	INFO	MWM_VM_DESTROY	VM %s has been terminated.	The named VM has finished its lifecycle and is now removed and added to the completed table.
0x100025a	USER	domain.lifecycle	INFO	MWM_VM_CANCEL	VM %s has been canceled.	The named VM has been canceled.
0x100025b	USER	domain.lifecycle	INFO	MWM_VM_END	VM %s has been ended.	The named VM has been canceled because it has exceeded its allocated walltime.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100025c	USER	domain.lifecycle	INFO	MWM_VM_MIGRATE_START	VM %s migration has started. (%s)	The named VM has started its migration. Additional information might be provided regarding source and destination nodes.
0x100025d	USER	domain.lifecycle	INFO	MWM_VM_MIGRATE_END	VM %s migration has finished. (%s)	The named VM has finished its migration. Additional information might be provided regarding source and destination nodes.
0x100025f	USER	domain.lifecycle	INFO	MWM_VM_MANUAL_MIGRATE_START	VM %s migration started manually. (%s)	The named VM migration has been started manually. Additional information might be provided regarding source and destination nodes.
0x1000260	USER	domain.lifecycle	INFO	MWM_VM_READY	VM %s is ready.	The named VM is ready. It has been linked to an internal

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						tracking job.
0x1002711	USER	system.moab	INFO	MWM_PARAMETER_SET_TO_VALUE_INFO	Parameter '%s' is set to '%s'.	A parameter was set to a specified value. This is usually accomplished via a configuration file.
0x1002741	USER	system.moab	INFO	MWM_SOCKET_EXCEPTION	Exception detected in select for socket %s.	The select() system call indicated an exception for this socket.
0x1002742	USER	system.moab	INFO	MWM_SOCKET_EXCEPTION_REASON	Exception identified as '%s' in select for socket %s.	The select() system call indicated an exception for this socket. It has been identified with an error ID by getsockopt().
0x1002748	USER	system.moab	INFO	MWM_MOAB_STARTED_ON_CORRECT_HOST	Server started on host '%s' %s.	Moab is started on either the primary or fallback server.
0x1002762	USER	system.moab	INFO	MWM_CONFIG_LINE_SUCCESSFUL	Configuration line '%s' successfully processed.	The line in the configuration file was processed without error.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1002935	USER	system.moab	INFO	MWM_ACTIVE_JOB_REMOVED_FROM_QUEUE	Active %s job %s has been removed from the queue, default to successful completion.	The job was removed from the indicated resource manager while it was still active. By default, it is assumed to complete successfully unless more information is available (i.e., ENABLEFAILUREFORPURGEDJOB).
0x1002936	USER	system.moab	INFO	MWM_INACTIVE_JOB_REMOVED_FROM_QUEUE	Inactive %s job %s has been removed from the queue, default to status 'canceled'.	The job was removed from the indicated resource manager while it was still active. By default, it is given status 'canceled' unless more information is available (i.e., ENABLEFAILUREFORPURGEDJOB).
0x1002937	USER	system.moab	INFO	MWM_RM_DOWN_SKIPPING_WORK	RM %s state is %s, skipping %s.	The specified resource manager is not in a good state. Certain actions might be skipped while

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						it is in this state.
0x100296a	USER	system.moab	INFO	MWM_ CANNOT_ RESUME_ JOB	Cannot resume job '%s' (%s).	Check the PBS server log to see reason for failure.
0x100296b	USER	system.moab	INFO	MWM_ CANNOT_ LOCATE_ RESOURCE	Cannot locate %s '%s'.	Unable to find the resource specified.
0x100296c	USER	system.moab	INFO	MWM_ CANNOT_ SET_JOB_ CLASS	Cannot set class on job '%s' to '%s' (%s).	The job could not be modified.
0x100296d	USER	system.moab	INFO	MWM_ NATIVE_ ACTION_ MISSING	%s action not specified for native interface. %s.	The native interface allows custom actions to be specified. No value was specified for this action.
0x100296e	USER	system.moab	INFO	MWM_ COMMAND_ SENT	Command sent to server.	A command was sent.
0x100296f	USER	system.moab	INFO	MWM_JOB_ MAXPREMPT	JOBMAXPREEMPTPERITERATION reached: %s of %s.	The maximum value was reached.
0x1002970	USER	system.moab	INFO	MWM_JOB_ CHANGED_ STATES	Job '%s' changed states from '%s' to '%s'.	The state changed.
0x1	US	syste	IN	MWM_JOB_	Job '%s' was	The job no

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
002971	ER	m.moab	FO	RELEASING_RESERVATION	requeued/rejected. Releasing reservation.	longer holds the reservation.
0x1002972	USER	system.moab	INFO	MWM_NODE_CHANGED_STATES	Node '%s' changed states from '%s' to '%s'.	The node state changed.
0x1002973	USER	system.moab	INFO	MWM_JOB_ACTION_SUCCESSFUL	Job '%s' successfully %s.	The job action completed.
0x1002974	USER	system.moab	INFO	MWM_ALLOC_TEMP_MEMORY	Cannot allocate temp memory for %s completed jobs.	The system might be low on memory.
0x1002975	USER	system.moab	INFO	MWM_ACTION_LAUNCHED	Action '%s' launched with message '%s'. PID = '%s'	Scheduler action is about to be executed.
0x1002976	USER	system.moab	INFO	MWM_JOB_ADJUSTMENT	Adjusting allocated %s to %s for job '%s'.	The value is being changed.
0x1002977	USER	system.moab	INFO	MWM_ALL_JOBS_LOADED	All jobs loaded.	The jobs have been loaded.
0x1002978	USER	system.moab	INFO	MWM_ALL_NODES_LOADED	All located non-native nodes loaded (%s).	The nodes have been loaded.
0x1002979	USER	system.moab	INFO	MWM_BACKFILL_POLICY_DISABLED	Backfill policy disabled.	The policy was disabled.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100297a	USER	system.moab	INFO	MWM_JOB_LOAD	Cannot load job '%s'.	The job failed to load.
0x100297b	USER	system.moab	INFO	MWM_CANNOT_CREATE_RSV	Cannot create reservation.	The request to create the given reservation has failed.
0x100297c	USER	system.moab	INFO	MWM_MODIFY_PARTITION	Cannot modify partition of running job '%s'.	Must wait until job completes.
0x100297f	USER	system.moab	INFO	MWM_CANNOT_CREATE_RSV_IN_PARTITION	Cannot create reservation for job '%s' in partition '%s'.	Job cannot be run on requested partition.
0x1002982	USER	system.moab	INFO	MWM_CLUSTER_QUERY_GETDATA	Cluster query getdata failed for native interface.	The resource manager might be down or unresponsive.
0x1002a0b	USER	system.moab	INFO	MWM_SENDING_CLIENT_COMMAND	Sending %s command: '%s'.	The specified command is being sent to the server.
0x1002a0e	USER	system.moab	INFO	MWM_SCHED_SHUTDOWN_REQUEST	The scheduler has received a user shutdown request.	The Moab scheduler has received a request to shut down. It will be processed as soon as possible.
0x1	US	syste	IN	MWM_	The scheduler has received	The Moab

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
002a0f	ER	m.moab	FO	SCHED_RECYCLE_REQUEST	a user recycle request.	scheduler has received a request to recycle. It will be processed as soon as possible.
0x1002a10	USER	system.moab	INFO	MWM_SCHED_PAUSE_DESCRIPTION	Scheduling will be disabled, cluster information will continue to be updated.	This is a description of what happens when the scheduler is paused.
0x1002a11	USER	system.moab	INFO	MWM_SCHED_STOP_TIMESTAMP	Scheduling will stop in %s at iteration %s.	This provides a log message of when the scheduler will stop.
0x1002a12	USER	system.moab	INFO	MWM_SCHED_RESUME_TIMESTAMP	Scheduling will resume in %s seconds.	This provides a log message of when the scheduler will resume.
0x1002a13	USER	system.moab	INFO	MWM_SCHED_RESTART_TIME_REACHED	Scheduler restart time reached (scheduler will restart).	The configured restart time was reached. (RESTARTINTERVAL or MEMREFRESHINTERVAL.
0x1002a14	USER	system.moab	INFO	MWM_SCHED_COMPLETE_SCHEDULING	Scheduling complete. Sleeping for %s seconds.	The scheduling portion of the iteration is complete. Additional jobs will not be

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						scheduled until the next iteration.
0x1002a17	USER	system.moab	INFO	MWM_ABOUT_TO_EXEC	About to exec() '%s'.	The process is about to be executed.
0x1002a18	USER	system.moab	INFO	MWM_JOB_ARRAY_CANCEL_POLICY	Sub-job %s exit code %s canceled job array %s with policy %s.	A job within an array job finished and, depending on its exit code and the policy in place, the entire array job might cancel.
0x1002a19	USER	system.moab	INFO	MWM_RESERVATION_COMPLETION_DELAYED	Reservation completion for job '%s' delayed from %s to %s.	The reservation end time is later than initially expected for this job.
0x1002a1b	USER	system.moab	INFO	MWM_VM_ORPHANED	VM '%s' successfully orphaned.	The VM has been separated from its tracking job.
0x1002a1c	USER	system.moab	INFO	MWM_VM_REPORTED_DESTROYED	VM '%s' reported destroyed via RM - removing VM.	The VM is no longer available from the resource manager, so it is being removed from the scheduler.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1002a1d	USER	system.moab	INFO	MWM_VM_STALE_REPORT	VM '%s' has not been reported in %s seconds.	The VM is no longer being reported from the resource manager. No action is currently being taken.
0x1002a1e	USER	system.moab	INFO	MWM_WIKI_KEYWORD_NOT_HANDLED	Wiki keyword '%s'(%s) not handled.	The keyword was not recognized, so it will be ignored.
0x1002a20	USER	system.moab	INFO	MWM_ROLLING_LOGFILE	Rolling logfile '%s' to '%s'.	The old logfile will be closed and logging will resume in the new file.
0x1002a23	USER	system.moab	INFO	MWM_NODE_LOCATED	Nodes located for job %s: %s of %s required (%s feasible).	List of nodes located for a specific job.
0x1002a2d	USER	system.moab	INFO	MWM_JOB_PAL_SET	Partition access list set to value: %s.	The partition access list (PAL) is set.
0x1002a2e	USER	system.moab	INFO	MWM_JOB_PREEMPTED_BY_JOB	Job %s preempted job %s - added idle resources (T: %s; N: %s; P: %s)/remaining (T: %s; N: %s; P: %s).	Job was preempted by another job.
0x1002a2f	USER	system.moab	INFO	MWM_JOB_CAN_START_WITHOUT_PREMEPTION	Job %s would start in %s without preemption (PC: %s).	Job can start without preemption.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1002a32	USER	system.moab	INFO	MWM_PARTITON_RESOURCES	Partition %s nodes/procs available %s/%s (%s jobs examined).	General partition information.
0x1002a33	USER	system.moab	INFO	MWM_RSV_OPERATION	Performing '%s' operation on reservation expression '%s' (%s matches).	This operation is caused by a mrsvctl command.
0x1002a34	USER	system.moab	INFO	MWM_PREEMPTING_JOBS	Preempting jobs to allow job %s to start - required resources T: %s; N: %s; P: %s.	Preempting jobs.
0x1002a35	USER	system.moab	INFO	MWM_MOABTRACKSUSPEND	Preempt usage tracking enabled (env).	Environment variable MOABTRACKSUSPEND set.
0x1002a36	USER	system.moab	INFO	MWM_JOB_MAX_PREEMPTEE_LIMIT	Single job max preemptee limit (%s) reached.	Max requirements exceeded on job.
0x1002a37	USER	system.moab	INFO	MWM_QUEUES_DETECTED	Queues detected: %s.	Resource manager found queues on cluster query update.
0x1002a38	USER	system.moab	INFO	MWM_JOB_START_TIME_CHANGED	Start time changed from %s to %s on job %s.	The job's start time was changed via the resource manager.
0x1002a39	USER	system.moab	INFO	MWM_STORING_CHECKPOINT_INFO	Storing object to checkpoint.	The object's state is being checkpointed.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1002a3a	USER	system.moab	INFO	MWM_PBS_DATA_UP_TO_DATE	PBS raw data already up to date.	The resource manager is already updated.
0x1002a3b	USER	system.moab	INFO	MWM_PBS_DATA_UPDATED	PBS data updated for iteration %s.	The resource manager is now updated.
0x1002a3c	USER	system.moab	INFO	MWM_STARTED_MESSAGE_QUEUE	Started message queue thread.	The message queue is now operational.
0x1002a3d	USER	system.moab	INFO	MWM_JOBS_SELECTED_IN_PARTITION	Total jobs selected in partition %s: %s/%s.	Identifies the selected jobs in a partition.
0x1002a3e	USER	system.moab	INFO	MWM_TASKS_LOCATED_FOR_JOB	Tasks located for job %s: %s of %s required (%s feasible).	Identifies the tasks available for a job.
0x1002a3f	USER	system.moab	INFO	MWM_CLIENT_REQUEST	Client requesting command '%s'.	Client requested command.
0x1002a40	USER	system.moab	INFO	MWM_REQUEST_TO_CANCEL_JOB	Request to cancel job '%s' sent, but could not confirm cancellation (pending response).	Client did not get a confirmation as expected.
0x1002a41	USER	system.moab	INFO	MWM_RESERVATION_NOT_ALLOWED_FOR_JOB	Reservation not allowed for job %s in %s.	Reservation not allowed in specified condition.
0x1	US	syste	IN	MWM_	Reserved job '%s' started.	Reserved job

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
002a42	ER	m.moab	FO	RESERVED_JOB_STARTED		started.
0x1002a43	USER	system.moab	INFO	MWM_RESOURCES_AVAILABLE_AFTER_SCHEDULING	Resources available after scheduling: N: %s P: %s.	Resources available after scheduling.
0x1002a44	USER	system.moab	INFO	MWM_RESTORING_DEFERRED_JOB	Restoring job '%s' from deferred state.	Restoring job from deferred state.
0x1002a45	USER	system.moab	INFO	MWM_RM_DUPLICATE_QUERY	RM %s already has a pending query - skipping get data query.	Duplicate queries cannot be performed simultaneously.
0x1002a46	USER	system.moab	INFO	MWM_RM_PEER_COMMAND	Sending peer server command to %s:%s (Cmd: %s, Requestor: %s, Key: %s...).	A command has been sent to a peer Moab grid server.
0x1002a47	USER	system.moab	INFO	MWM_SET_ATTRIBUTE_ON_NODE	Setting %s on node %s to %s.	A command has been sent to a peer Moab grid server.
0x1002a48	USER	system.moab	INFO	MWM_SET_ATTRIBUTE_ON_JOB	Setting %s on job %s to %s (%s).	A command has been sent to a peer Moab grid server.
0x1002a49	USER	system.moab	INFO	MWM_INVALID_STORAGE_DATA	Storage data from MWS RM (%s) is not a valid object.	Invalid object from Storage data.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a73c	ADMIN	system.moab	INFO	MWM_CANNOT_STAT_FILE_INFO	Cannot stat file '%s', errno: %s (%s).	The stat() system call failed. This is not always significant as it is sometimes used to test the existence of a file that may or may not be there. Use the errno and associated message to determine possible causes.
0x100a743	ADMIN	system.moab	INFO	MWM_FAILED_SELECT	Select for socket %s failed, errno: %s (%s).	The select() system call failed. Use the errno and associated message to determine possible causes.
0x100a744	ADMIN	system.moab	INFO	MWM_SELECT_TIMEOUT	Select for socket %s timed out after %s seconds with no valid descriptors.	The select() system call timed out. This may or may not be an error. Check MTU.
0x100a75d	ADMIN	system.moab	INFO	MWM_CONFIG_VALUE_OUT_OF_RANGE	Configuration parameter '%s' has an invalid value '%s'. Range is limited by %s.	Check the line in the configuration file for the attribute.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a767	ADMIN	system.moab	INFO	MWM_NO_MONGOSERVER_SPECIFIED	Moab is built to use Mongo, but no MONGOSERVER is specified.	Cannot connect to the Mongo server since the MONGOSERVER parameter was unspecified. Add MONGOSERVER parameter to moab.cfg.
0x100a76d	ADMIN	system.moab	INFO	MWM_REMOVING_OBJECT_FROM_MONGO	Removing object '%s' from Mongo DB '%s'.	The object is being removed from the database.
0x100a789	ADMIN	system.moab	INFO	MWM_UNABLE_TO_ALLOCATE_NODES_FOR_RSV	Cannot allocate nodes for reservation '%s'. (%s)	Cannot allocate a node list that matches the requirements for this reservation. This might not be serious since multiple passes can occur.
0x100a8ab	ADMIN	system.moab	INFO	MWM_VM_EXCEED_TTL	VM '%s' has reached TTL (%s). Must be removed manually.	The given VM has reached its time to live.
0x100a8e1	ADMIN	system.moab	INFO	MWM_PLUGIN_LOADED_SUCCESS	Successfully loaded NodeAllocation plugin '%s' for partition '%s'.	A NodeAllocation plug-in was loaded without error.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a918	ADMIN	system.moab	INFO	MWM_JOB_NO_QUEUETIME	No QueueTime has been specified for job.	Configure the job with a queue time.
0x100a922	ADMIN	system.moab	INFO	MWM_FIND_PEER	Cannot find client peer for job %s (Name: %s).	The resource manager cannot be located.
0x100a95b	ADMIN	system.moab	INFO	MWM_RSV_FULL	Full reservation '%s' reserved %s procs in partition '%s' to start in %s at (%s).	The full reservation has been reserved.
0x100a97d	ADMIN	system.moab	INFO	MWM_RSV_PREREQ_JOB	Cannot create reservation for prerequisite job '%s'.	Could not obtain a reservation for this job.
0x100a97e	ADMIN	system.moab	INFO	MWM_ANNOTATE_JOB	Cannot annotate job '%s' with message '%s'.	Unable to modify the job with the annotation.
0x100a980	ADMIN	system.moab	INFO	MWM_UPDATE_JOB	Cannot update job '%s'.	The update on the job from XML failed.
0x100a981	ADMIN	system.moab	INFO	MWM_REMAP_CLASS	Cannot remap class for RM job '%s' (%s).	Unable to modify the job with the new class.
0x100a983	ADMIN	system.moab	INFO	MWM_COMPLETING_JOB	Completing job '%s'.	The job finished.
0x100a984	ADMIN	system.moab	INFO	MWM_JOBS_DETECTED	There were %s %s jobs detected on RM '%s'.	The resource manager reported these

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						jobs.
0x100a985	ADMIN	system.moab	INFO	MWM_SUSPEND_JOB	Cannot suspend job '%s' (%s).	Check the PBS server log to see reason of failure.
0x100a986	ADMIN	system.moab	INFO	MWM_STALE_PARTITION	Attempting to remove stale partition for completed job '%s'.	About to perform the stated operation.
0x100a987	ADMIN	system.moab	INFO	MWM_STALE_PARTITION_SUCCESS	Successfully removed stale partition for completed job '%s'.	Successfully performed the stated operation.
0x100a988	ADMIN	system.moab	INFO	MWM_CANCEL_NOQUEUE_JOB	Canceling No-queue job '%s'.	About to perform the stated operation.
0x100a989	ADMIN	system.moab	INFO	MWM_SIGNAL_JOB	Cannot signal job %s' (%s).	The resource manager did not respond to the signal request.
0x100a98a	ADMIN	system.moab	INFO	MWM_RSV_JOB_CREDS	Cannot set up reservation job credentials.	The user, account, or group credentials might not be valid.
0x100a98b	ADMIN	system.moab	INFO	MWM_CP_CORRUPT_NODE_LINE	Corrupt node line detected (%s).	The line does not contain the correct syntax for a checkpoint.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a98c	ADMIN	system.moab	INFO	MWM_EVALUATING_RSV	Evaluating reservation '%s'.	About to perform the stated operation.
0x100a98d	ADMIN	system.moab	INFO	MWM_EXPIRING_CHECKPOINT_DATA	Expiring checkpoint data for %s '%s'. Not updated in %s.	The object's checkpoint data has expired.
0x100a98e	ADMIN	system.moab	INFO	MWM_JOB_PREVIOUSLY_REMOVED	Job '%s' was previously removed.	The job has already been removed.
0x100a98f	ADMIN	system.moab	INFO	MWM_JOB_STARTED_BY_USER	Job '%s' was started by user '%s'.	The job started.
0x100a990	ADMIN	system.moab	INFO	MWM_JOB_NOT_STARTED_BY_USER	Job '%s' could not be started by user '%s' (%s).	The job could not be started.
0x100a991	ADMIN	system.moab	INFO	MWM_RM_JOB_NOT_STARTED	Job '%s' could not be started with %s RM '%s' (%s).	The job could not be started.
0x100a992	ADMIN	system.moab	INFO	MWM_JOB_CANCELED_EXTERNALLY	Job '%s' appears to have been canceled externally.	The job was canceled.
0x100a993	ADMIN	system.moab	INFO	MWM_JOB_COMPLETED_SINGLE_ITERATION	Job '%s' appears to have been started and completed in a single iteration.	The job completed.
0x100a994	ADMIN	system.moab	INFO	MWM_JOB_PROCESSING_	Job processing completed.	The jobs have been processed.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
				COMPLETED		
0x100a995	ADMIN	system.moab	INFO	MWM_PROCESSING_JOB	Processing job '%s' in state '%s'.	Processing a single job.
0x100a996	ADMIN	system.moab	INFO	MWM_JOB_SUSPENDED	Job '%s' suspended through %s RM.	The resource manager suspended the job.
0x100a997	ADMIN	system.moab	INFO	MWM_JOB_RESUMED	Job '%s' resumed through %s RM.	The resource manager resumed the job.
0x100a998	ADMIN	system.moab	INFO	MWM_JOB_FEASIBILITY_CHECK_DISABLED	Job feasibility check disabled (env).	This feature has been disabled.
0x100a999	ADMIN	system.moab	INFO	MWM_JOB_USAGE_SENT	Job usage sent for job '%s'.	The usage sent as feedback to user.
0x100a99a	ADMIN	system.moab	INFO	MWM_LOADING_JOBS	Loading %s job(s).	The jobs are about to be loaded.
0x100a99b	ADMIN	system.moab	INFO	MWM_LOADING_NODE_RECORDS	Loading %s node record(s).	The node records are about to be loaded.
0x100a99c	ADMIN	system.moab	INFO	MWM_LOADED_WORKLOAD_BUFFER	Loaded %s workload buffer (%s bytes), processing jobs.	The workload buffer was loaded.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a99d	ADMIN	system.moab	INFO	MWM_JOB_REJECTED_INFINITE_WALLTIME	Job '%s' rejected (requested infinite walltime).	Jobs must have a walltime limit.
0x100a99e	ADMIN	system.moab	INFO	MWM_JOB_REJECTED_PARTITION	Job '%s' rejected in partition %s (exceeds maximum task size: %s > %s).	Adjust JOBMAXTASKCOUNT in the configuration file.
0x100a99f	ADMIN	system.moab	INFO	MWM_JOB_ALREADY_EXISTS	Job '%s' already exists but is not a duplicate.	The ID of the job matched a completed job.
0x100a9a0	ADMIN	system.moab	INFO	MWM_JOB_ALREADY_BATCH_HOLD	Job '%s' is already on batch hold.	Trying to place a job on hold that is already in that state.
0x100a9a1	ADMIN	system.moab	INFO	MWM_JOB_REQUESTS_RSV	Job '%s' requests reservation '%s' (not deferring).	The job is requesting the reservation.
0x100a9a2	ADMIN	system.moab	INFO	MWM_RM_CONNECTION_FAILED	Connection to RM '%s' failed. Not deferring job '%s' (Reason: %s).	Refer to the reason message.
0x100a9a3	ADMIN	system.moab	INFO	MWM_DEFER_DISABLED	Defer disabled.	The job cannot be deferred.
0x100a9a4	ADMIN	system.moab	INFO	MWM_MWS_CLUSTER_QUERY	Cluster query retrieval failed for MWS RM '%s'.	The resource manager did not respond to the request.
0x100a9a5	ADMIN	system.moab	INFO	MWM_JOB_INVALID_PARTITION	Job '%s' specifies an invalid partition.	The job must reference a valid partition.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a9a6	ADMIN	system.moab	INFO	MWM_JOB_INVALID_QOS	Cannot set QoS on job '%s' to '%s' - invalid QoS.	The job must use a valid QoS.
0x100a9a7	ADMIN	system.moab	INFO	MWM_JOB_INVALID_ACCOUNT	Cannot set account on job '%s' to '%s' - invalid account (%s).	The job must use a valid account.
0x100a9a8	ADMIN	system.moab	INFO	MWM_CHECKING_IDLE_JOB	Checking idle job '%s' (priority: %s) partition %s.	Checking the job.
0x100a9a9	ADMIN	system.moab	INFO	MWM_CHECKING_SUSPENDED_JOB	Checking suspended job '%s' (priority: %s).	Checking the job.
0x100a9aa	ADMIN	system.moab	INFO	MWM_CHECKPOINT_TEST_ENABLED	Checkpoint test enabled (env).	The feature has been enabled with an environment variable.
0x100a9ab	ADMIN	system.moab	INFO	MWM_ADD_NODE_FAILED	Could not add node because MNodeAdd failed.	The node could not be added to the object.
0x100a9ac	ADMIN	system.moab	INFO	MWM_ATTEMPTING_RESERVATION	Attempting reservation of %s procs in %s for %s.	The scheduler will try to make the reservation.
0x100a9ad	ADMIN	system.moab	INFO	MWM_FAIRSHARE_INTERVAL	Fairshare rolled to interval %s.	The interval has changed.
0x100a9ae	ADMIN	system.moab	INFO	MWM_INVALID_ARCHITECT	Invalid architecture.	The architecture is not a valid

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
				URE		value.
0x100a9af	ADMIN	system.moab	INFO	MWM_INVALID_PSEUDOJOB	Invalid pseudo-job.	The pseudo-job is not a valid value.
0x100a9b0	ADMIN	system.moab	INFO	MWM_HOLD_TYPE	Hold type '%s' selected.	The given hold type was specified.
0x100a9b1	ADMIN	system.moab	INFO	MWM_MESSAGE_SENT	Message sent to server.	The message was sent.
0x100a9b2	ADMIN	system.moab	INFO	MWM_JOB_LOCATED	Located job '%s' in partition '%s' reserved to start %s.	The specified job has been located.
0x100a9b3	ADMIN	system.moab	INFO	MWM_TOTAL_JOBS_DETECTED	Total jobs detected: %s.	Number of counted jobs returned from the workload query.
0x100a9b4	ADMIN	system.moab	INFO	MWM_NO_WORKLOAD_DETECTED	No workload reported by any RM.	No jobs were reported across all the resource manager queries.
0x100a9b5	ADMIN	system.moab	INFO	MWM_LOADING_JOB	Loading job '%s' in state '%s' (%s bytes).	The job is being loaded.
0x100a9b6	ADMIN	system.moab	INFO	MWM_JOB_START_REJECTED	Local constraints rejected the starting of job '%s'.	The job cannot start.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a9b7	ADMIN	system.moab	INFO	MWM_INVALID_STAT_TYPE	Invalid stat type '%s' requested.	Not a valid value.
0x100a9b8	ADMIN	system.moab	INFO	MWM_ORPHAN_PARTITION	Creating temporary job to process orphan partition '%s' for job '%s'.	The job was not found in active or completed job tables.
0x100a9b9	ADMIN	system.moab	INFO	MWM_DISABLING_ACTION_PROGRAM	Disabling action program '%s'.	An invalid action program was requested.
0x100a9ba	ADMIN	system.moab	INFO	MWM_DISABLING_JOB_FB_PROGRAM	Disabling job feedback program '%s' (%s).	An invalid job feedback program was requested. See the parameter FEEDBACKPROGRAM.
0x100a9bb	ADMIN	system.moab	INFO	MWM_CP_RESTART_STATE_IGNORED	Checkpoint restart state '%s' ignored.	The restart state specified is being ignored.
0x100a9bc	ADMIN	system.moab	INFO	MWM_CP_RESTART_STATE_SUCCESS	Starting scheduler with checkpoint restart state '%s'.	The restart state specified is being used.
0x100a9bd	ADMIN	system.moab	INFO	MWM_DESTROYING_NODE	Destroying node '%s'.	The specified node is being destroyed.
0x100a9be	ADMIN	system.moab	INFO	MWM_IGNORING_NODE	Ignoring node '%s'.	The specified node is being ignored.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a9c0	ADMIN	system.moab	INFO	MWM_CANNOT_ADJUST_JOB_HOLDS	Cannot adjust holds on remote peer for job '%s' (%s).	Unable to modify the job.
0x100a9c1	ADMIN	system.moab	INFO	MWM_JOB_CANNOT_CREATE_RESERVATION	Cannot create reservation for job '%s' (previously reserved to start in %s)).	Failed to create reservation for job.
0x100a9c2	ADMIN	system.moab	INFO	MWM_TRIGGER_LOAD_OUTPUT	Cannot load output data for trigger '%s' (File: %s).	The file might not exist or might be inaccessible.
0x100a9c3	ADMIN	system.moab	INFO	MWM_PBS_SERVER_CONNECT	Connected to PBS server %s:%s on sd %s.	Connection established.
0x100a9c4	ADMIN	system.moab	INFO	MWM_NO_JOB_DATA	No job data was sent by %s RM.	The data sent by the resource manager did not contain job information.
0x100a9c5	ADMIN	system.moab	INFO	MWM_JOB_RESUMED_WITH_PROCS	Job '%s' resumed on %s processors.	The resource manager resumed the job.
0x100a9c6	ADMIN	system.moab	INFO	MWM_JOB_SIGNALED	Job %s' successfully signaled (action: %s, signal: %s).	The job responded to the signal request.
0x100a9c7	ADMIN	system.moab	INFO	MWM_JOB_CANCELED_RM	Job '%s' canceled through %s RM.	The job was canceled.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a9c8	ADMIN	system.moab	INFO	MWM_JOB_ASSIGNED_DEFAULT_GROUP	Job '%s' assigned default group '%s'.	The job was modified.
0x100a9c9	ADMIN	system.moab	INFO	MWM_FILE_EXECUTE_PERMISSION	File '%s' does not have user execute permission (st_mode = %s).	The permissions must be modified.
0x100a9ca	ADMIN	system.moab	INFO	MWM_INSUFFICIENT_PREEMPT_JOBS	Inadequate preempt jobs (%s) located for %s job (P: %s of %s, N: %s of %s).	Not enough jobs could be preempted.
0x100a9cb	ADMIN	system.moab	INFO	MWM_READ_STAT_INDEX	Cannot read stat index for location %s:%s:%s.	The checkpoint did not have the stat information.
0x100a9cc	ADMIN	system.moab	INFO	MWM_BACKFULL_JOB_PREEMPT	Backfill job '%s' no longer preemptible (%s > %s) in partition '%s'.	The job cannot be preempted.
0x100a9cd	ADMIN	system.moab	INFO	MWM_STARTTIME_UNAVAILABLE	Cannot obtain desired starttime (%s != %s).	The job cannot be adjusted to the given start time.
0x100a9ce	ADMIN	system.moab	INFO	MWM_STARTTIME_ADJUSTED	Timeframe for reservation %s adjusted forward by %s seconds.	The reservation has been adjusted to the given start time.
0x100a	ADMI	system.mo	INF	MWM_RESERVATI	Time: %s RollbackOffset: %s RsvStartTime: %s	The reservation is

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
9cf	N	ab	O	ON_ ROLLBACK	RsvDuration %s.	being considered for rollback.
0x100a9d0	ADMIN	system.moab	INFO	MWM_ RESERVATION_NOT_ REQUIRED	Reservation '%s' not required for specified period.	The reservation is not required for this time period.
0x100a9d1	ADMIN	system.moab	INFO	MWM_RM_ INTERFACE_ RECOVERED	The interface for RM '%s' has been recovered.	A previously corrupt interface is now working.
0x100a9d2	ADMIN	system.moab	INFO	MWM_NTR_ JOB_FOUND	Found an NTR (next to run) job - stopping idle job scheduling.	The job will now be run.
0x100a9d3	ADMIN	system.moab	INFO	MWM_GRES_ KEYWORD_ NO_VALUE	GRes keyword '%s' passed in with no value.	A value must be specified.
0x100a9d4	ADMIN	system.moab	INFO	MWM_JOB_ EXTENSION_ STRING	Job '%s' has invalid extension string - '%s'.	The system is unable to process the string.
0x100a9d5	ADMIN	system.moab	INFO	MWM_JOB_ PROCESS_ FAILURE	Job '%s' is invalid. It cannot be processed (%s).	There was an error loading the job. It will be rejected.
0x100a9d6	ADMIN	system.moab	INFO	MWM_JOB_ MODIFIED_ RM	Job '%s' has been modified through %s RM.	The job was modified.
0x100a9d7	ADMIN	system.moab	INFO	MWM_IDLE_ BACKLOG_ SIZE	Idle backlog: %s seconds (%s hours).	The idle backlog status is given.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a9d8	ADMIN	system.moab	INFO	MWM_SET_RESOURCES	Inadequate resources found in any set (%s < %s).	None of the node sets have the resources needed.
0x100a9d9	ADMIN	system.moab	INFO	MWM_JOB_PREEMPTING_JOB	Job %s preempting job %s (statemtime: %s) (preempted this iteration: %s).	One job preempted another.
0x100a9da	ADMIN	system.moab	INFO	MWM_UPDATE_SCHEDULER_STATS	Iteration: %s; scheduling time: %s seconds.	Normal statistics update.
0x100a9db	ADMIN	system.moab	INFO	MWM_JOB_STARTED_RM	Job '%s' started through %s RM on %s procs.	The job has started.
0x100a9dc	ADMIN	system.moab	INFO	MWM_JOB_DELAY	Job delay: %s; reservation retry time: %s (StateDelayNC: %s; JobRsvDelayNC: %s).	The job has been delayed.
0x100a9dd	ADMIN	system.moab	INFO	MWM_JOB_COMPLETED	Job '%s' completed. X: %s; T: %s; PS: %s; A: %s (RM: %s/%s).	The job completed.
0x100a9de	ADMIN	system.moab	INFO	MWM_JOB_RESERVED_TASKS	Job '%s' reserved %s tasks (partition %s) to start in %s on %s (WC: %s).	The job has reserved the tasks.
0x100a9df	ADMIN	system.moab	INFO	MWM_EVENT_INTERFACE_ENABLED	Event interface enabled for wiki RM %s on port %s.	The interface is now functional.
0x100a9e0	ADMIN	system.moab	INFO	MWM_RM_RESOURCES_DETECTED	There were %s %s resources detected on RM '%s'.	The given resources were found.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a9e3	ADMIN	system.moab	INFO	MWM_EXTENDING_RESERVATION	Extending reservation by %s seconds (trigger still active).	The reservation is being extended.
0x100a9e4	ADMIN	system.moab	INFO	MWM_EXTENDING_RESERVATION_OVERRUN_JOB	Extending reservation for overrun job '%s' by %s seconds.	The reservation is being extended.
0x100a9e5	ADMIN	system.moab	INFO	MWM_JOB_LOCATED_BESTFIT	Located bestfit job '%s' (size: %s; duration: %s).	Backfill found a job that best fits the available resources.
0x100a9e6	ADMIN	system.moab	INFO	MWM_JOB_BEST_PARTITION	The best partition for job '%s' is '%s'.	Backfill found a job that best fits the available resources.
0x100a9e7	ADMIN	system.moab	INFO	MWM_CPA_PARTITION_DESTROY	Destroying CPA partition'%s' for job '%s' with cookie %s (%s).	The partition is being destroyed.
0x100a9e8	ADMIN	system.moab	INFO	MWM_RESOURCES_LOCATED	Located resources for %s tasks (%s) in best partition '%s' for job '%s' at time offset %s.	The listed resources have been located.
0x100a9e9	ADMIN	system.moab	INFO	MWM_MINIMUM_EFFICIENCY_REACHED	Minimum efficiency reached (%s percent) on iteration %s.	The threshold has been reached.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a9ea	ADMIN	system.moab	INFO	MWM_JOB_START_PARTITION	Cannot start job '%s' in partition '%s' (scheduler mode: %s).	The job could not be started.
0x100a9eb	ADMIN	system.moab	INFO	MWM_JOB_FEASIBLE_NODES	Inadequate feasible nodes found for job '%s':%s in partition '%s' (%s < %s).	The job could not be scheduled.
0x100a9ec	ADMIN	system.moab	INFO	MWM_JOB_LOADED	Job '%s' loaded: TC=%s UGC=%s,%s,%s WC=%s ST=%s %s %s.	The job was loaded.
0x100a9ed	ADMIN	system.moab	INFO	MWM_JOB_RSV_CREATE	Cannot create new reservation for job %s (shape[%s] %s).	Check the reservation time, nodes, and account.
0x100a9ee	ADMIN	system.moab	INFO	MWM_JOB_LOCATE_NODES	Cannot locate nodes for job '%s' req[%s] (%s additional needed).	Not enough nodes are available to run the job.
0x100a9ef	ADMIN	system.moab	INFO	MWM_JOB_START_RM_DISABLED	Cannot start job '%s' since RM '%s' is disabled.	Not enough nodes are available to run the job.
0x100a9f0	ADMIN	system.moab	INFO	MWM_JOB_START_RESERVE_TIME	Cannot start job '%s' reserve time in %s.	The time to schedule has already arrived.
0x100a9f1	ADMIN	system.moab	INFO	MWM_ERROR_IN_EXE_STDERR	Error detected in '%s' due to presence of the word 'ERROR' in stderr (%s).	The executable failed.
0x100a9f2	ADMIN	system.moab	INFO	MWM_ERROR_IN_STDERR	Error detected due to presence of the word 'ERROR' in stderr.	The child process failed.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100a9f3	ADMIN	system.moab	INFO	MWM_CHECKJOB_STATE	Job '%s' State: %s Expected State: %s QueueTime: %s.	The job is in the listed state. The expected state might not be the same.
0x100a9f4	ADMIN	system.moab	INFO	MWM_JOB_NODELIST	Cannot obtain nodelist for job '%s':%s in range %s.	The nodes are not available.
0x100a9f5	ADMIN	system.moab	INFO	MWM_JOB_RESUME	Job '%s' cannot be resumed since allocated nodes are not available (node '%s' state '%s').	The resource manager resumed the job.
0x100a9f6	ADMIN	system.moab	INFO	MWM_CLEARING_EXPIRED_RESERVATION	Clearing expired %s reservation '%s' on iteration %s (start: %s end: %s).	The reservation has expired.
0x100a9f7	ADMIN	system.moab	INFO	MWM_CPA_RETRY	CPA retry detected - will re-attempt partition creation in 2 seconds.	The partition may be created.
0x100a9f9	ADMIN	system.moab	INFO	MWM_JOBS_STARTED	There were %s %s jobs started in partition '%s' on iteration %s.	The jobs were started.
0x100a9fa	ADMIN	system.moab	INFO	MWM_TASKS_ALLOCATED	There were %s of %s tasks allocated for job '%s':%s.	The tasks were allocated.
0x100a9fb	ADMIN	system.moab	INFO	MWM_CLASSES_DETECTED	There were %s %s classes/queues detected on RM '%s'.	The classes were detected.
0x100a9fd	ADMIN	system.moab	INFO	MWM_JOB_DELAYED_RSV	Delayed reservation detected for reserved job '%s' (%s seconds)	Attempting to fit the job into the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
					attempting squeeze.	reservation.
0x100a9fe	ADMIN	system.moab	INFO	MWM_DUMPING_RESERVATIONS	Dumping reservations on iteration %s.	All the reservations will be dumped to the log.
0x100a9ff	ADMIN	system.moab	INFO	MWM_ALLOCPARTITION_MISSING	ALLOCPARTITION missing from completed job '%s' - restoring variable with value '%s'.	The value is being substituted.
0x100aa00	ADMIN	system.moab	INFO	MWM_RECEIVED_NODELIST	Received nodelist through %s RM.	The nodelist was received.
0x100aa01	ADMIN	system.moab	INFO	MWM_SERVICE_REQUEST_FROM_HOST	Received service request from host '%s'.	The request was received.
0x100aa02	ADMIN	system.moab	INFO	MWM_RECEIVED_WORKLOAD	Received workload info through %s RM '%s' (%s bytes).	The workload was received.
0x100aa03	ADMIN	system.moab	INFO	MWM_RSV_REMOVED_FROM_CACHE	Removing reservation '%s' from cache.	The cached reservation is being removed.
0x100aa04	ADMIN	system.moab	INFO	MWM_RECOVER_READ_SOCKET	RECOVER: attempting to read socket connection.	The recovery function is attempting to communicate via sockets.
0x100aa05	ADMIN	system.moab	INFO	MWM_GREEDY_BACKFILL	Improved list found by greedy backfill in %s searches (utility: %s; processors available: %s).	The object is being removed from the database.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100aa06	ADMIN	system.moab	INFO	MWM_RESERVATION_NAME_AND_GROUP	Name='%s' RsvGroup='%s'.	The object is being removed from the database.
0x100aa15	ADMIN	system.moab	INFO	MWM_TRIGGERS_DISABLED	Triggers disabled. %s.	Triggers are disabled. This message indicates when this flag is being set, and when an action is being skipped because the flag is set.
0x100aa16	ADMIN	system.moab	INFO	MWM_USER_NOT_AUTHORIZED	User %s is not authorized to %s.	This user does not have permissions to accomplish the listed task.
0x100aa1f	ADMIN	system.moab	INFO	MWM_SUCCESSFULLY_OPENED_SOCKET	Opened service socket on port %s.	A socket was successfully opened listening on the remote port.
0x100aa21	ADMIN	system.moab	INFO	MWM_NO_CHECKPOINT_INFO	No checkpoint information available for '%s'.	Checkpoint information was not available.
0x100aa22	ADMIN	system.moab	INFO	MWM_NODE_INDEX_TABLE_ENABLED	Node index table enabled.	Enabled by environment variable: MOABUSENODEINDEX.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100aa24	ADMIN	system.moab	INFO	MWM_UNKNOWN_NODE_SLOT	Node slot not yet set on node '%s'.	Delaying setting rack until slot is known.
0x100aa25	ADMIN	system.moab	INFO	MWM_NODE_STATUS	Node '%s' status: state='%s' rsvlist='%s' joblist='%s'.	General node status.
0x100aa26	ADMIN	system.moab	INFO	MWM_NO_JOBS_IN_QUEUE	No jobs in queue.	There were no jobs in the scheduler queue indicating the scheduler has nothing to process.
0x100aa27	ADMIN	system.moab	INFO	MWM_NO_NODE_DATA	No node data sent by %s RM.	The resource manager did not receive any node data in cluster query.
0x100aa28	ADMIN	system.moab	INFO	MWM_NO_PREEMPTIBLE_RESOURCES	No preemptible resources found for job %s (tc: %s; class: '%s'; qos: %s; priority: %s; partition %s.	Indicates the scheduler could not find any jobs for preemption.
0x100aa29	ADMIN	system.moab	INFO	MWM_NO_PRIORITY_RESERVATION_CREATED_FOR_POLICY	No priority reservations created for policy '%s' for job %s.	Job reservation for a specific policy was unable to be created.
0x100aa2a	ADMIN	system.moab	INFO	MWM_NO_QUEUES_DETECTED	No queues detected.	Resource manager attempted to

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						obtain queue information. Check resource manager for configured queues.
0x100aa2b	ADMIN	system.moab	INFO	MWM_NOT_ADDING_RM	Not adding RM '%s'.	The partition is not adding the specified resource manager. This situation is most common in grid configurations where resource manager names are similar.
0x100aa30	ADMIN	system.moab	INFO	MWM_ORPHAN_PARTITION_REPORTED_FOR_JOB	Orphan partition %s reported for job %s. %s.	The resource manager reported the partition as orphaned.
0x100aa31	ADMIN	system.moab	INFO	MWM_PARAMETER_CANNOT_BE_CHANGED	Parameter '%s' cannot be changed while Moab is running.	Configuration file must be changed, and Moab must be restarted.
0x100aa4c	ADMIN	system.moab	INFO	MWM_NODE_REMOVED	Dynamic node '%s' is being removed. RequestID = '%s', TTL = %s, Reason = '%s'	A dynamic node was removed due to one of the following reasons: TTL expiration, mnodectl

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						deletion, node idle purge time, not reported in most recent cluster query.
0x100aa4d	ADMIN	system.moab	INFO	MWM_NODE_ADDED	Node '%s' was newly reported in the last cluster query. RequestID = '%s', TTL = %s	A node was newly reported in the last cluster query.
0x100aa4e	ADMIN	system.moab	INFO	MWM_NODE_IGNORED_TTL	Ignored dynamic node '%s' reported in cluster query with expired TTL. RequestID = '%s', TTL = %s	A dynamic node was reported in the last cluster query with an expired TTL - ignoring.
0x100aa4f	ADMIN	system.moab	INFO	MWM_NODE_IDLE_PURGE_TIME_EXCEEDED	dynamic node '%s' exceeded node idle purge time. RequestID = '%s'	dynamic node exceeded node idle purge time
0x100aa50	ADMIN	system.moab	INFO	MWM_NODE_UNREMOVED	Dynamic node '%s' is being re-added	A dynamic node was re-added
0x100aa51	ADMIN	system.moab	INFO	MWM_NODE_TTL_MODIFIED	Node '%s' TTL modified. RequestID = '%s', Old TTL = %s, New TTL = %s	The node TTL has been modified.
0x100aa52	ADMIN	system.moab	INFO	MWM_NODE_REQUESTID_MODIFIED	Node '%s' RequestID modified. Old RequestID = '%s', New RequestID = '%s'	The node RequestID has been modified.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x100aa63	ADMIN	system.moab	INFO	MWM_NO_MONGOREPLICASET_SPECIFIED	A set (replicaset) of Mongo servers (comma separated list) was given in MONGOSERVER, but MONGOREPLICASETNAME is undefined.	Cannot connect to the Mongo server. A set (replicaset) of Mongo servers (comma separated list) was defined by MONGOSERVER, but MONGOREPLICASETNAME is undefined. Add MONGOREPLICASETNAME parameter to moab.cfg.
0x100c3e8	INTERNAL	telemetry.timing	INFO	MWM_PERFORMANCE_TIMER	File:%s,Function:%s,Line:%s,Pid:%s,Duration:%s	Microsecond performance timer.
0x100c3e9	INTERNAL	telemetry.timing	INFO	MWM_FUNCTION_START	File:%s,Function:%s,Line:%s,Pid:%s,TimeIn:%s	Function start time.
0x100c3ea	INTERNAL	telemetry.timing	INFO	MWM_FUNCTION_END	File:%s,Function:%s,Line:%s,Pid:%s,TimeOut:%s,Duration:%s	Function end time.
0x100c3eb	INTERNAL	telemetry.output	INFO	MWM_FUNCTION_OUTPUT	File:%s,Function:%s,Line:%s,Pid:%s	Function output.
0x100c3f2	INTERNA	telemetry.cpu	INFO	MWM_CPU_MONITOR	Description:%s,Pid:%s,Ticks:%s,Percent:%s	CPU Utilization.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
	L					
0x100c3f3	INTERNAL	telemetry.stats	INFO	MWM_JOB_INFORMATION	Phase:%s,Count:%s	Job Process Information.
0x100c3fc	INTERNAL	telemetry.stats	INFO	MWM_SCHED_ITERATION	Schedule Phase:%s,Milliseconds:%s	Last Schedule Iteration.
0x100e72f	INTERNAL	system.moab	INFO	MWM_SOCKET_REMOTE_DISCONNECT	Reading from a socket failed. It appears the client disconnected, errno: %s (%s).	The recv() system call failed. Use the errno and associated message to determine possible causes.
0x100e7f0	INTERNAL	system.moab	INFO	MWM_VM_LINKED_TO_NEW_TRACKING_JOB	Setting VMTracking job for VM '%s' to job '%s.	A VM is associated with a tracking job.
0x110001f7	USER	domain.lifecycle	WARN	MWM_TRIG_FAILURE	Trigger %s has failed.	The named trigger has finished its action, but it returned with a failure status.
0x1100025e	USER	domain.lifecycle	WARN	MWM_VM_MIGRATE_END_ERROR	VM %s migration has finished with an error: (%s)	The named VM has finished its migration. There was a problem

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						during the migration. Additional information might be provided regarding the error specifics.
0x11000261	USER	domain.lifecycle	WARN	MWM_VM_MIGRATE_SUBMIT	Failed to submit VM migration job for VM %s. (%s)	Failed to submit VM Migration job.
0x11000262	USER	domain.lifecycle	WARN	MWM_VM_NO_FEASIBLE_NODES	Failed to find a feasible node/hypervisor on which to run VM %s. Check setup job %s for details.	The named VM has been submitted, but no node/hypervisor meets all requirements.
0x1100384	USER	system.moab	WARN	MWM_VM_LICENSE_ERROR	There is an error with the Moab license: (%s)	There was a licensing error. Additional information might be provided regarding the error specifics.
0x110284a	USER	system.moab	WARN	MWM_BAD_COMMANDLINE_FLAG	Unexpected flag detected: '%s'.	The command line syntax that was received contains an invalid flag. Check the documentation and retry.
0x1	US	syste	W	MWM_	Ignoring incorrect	Valid range is

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
1002858	ER	m.moab	ARN	NODESETMAXUSAGE_FAILURE	NODESETMAXUSAGE value '%s'.	from 0.0 to 1.0 inclusive.
0x11002883	USER	system.moab	WARN	MWM_CANNOT_DESTROY_STATIC_RM	Trigger cannot destroy static RM.	A trigger cannot destroy a static resource manager. Refer to trigger 'destroy'.
0x11002966	USER	system.moab	WARN	MWM_RM_JOB_SUBMIT_FAILURE	RM %s job submit failed: %s.	Error while submitting the job to the resource manager.
0x11002a0c	USER	system.moab	WARN	MWM_CANNOT_MODIFY_RM_JOB	Cannot modify %s for RM job %s - '%s'.	The listed attribute of the job could not be changed.
0x11004004	POWER_USER	system.moab	WARN	MWM_TESTING_WARNING	Testing with argument1: %s. and argument2: %s and argument3: %s	Internal error for testing diagnostics.
0x11008199	ADMIN	system.moab	WARN	MWM_SCHED_SET_COUNTER	%s cannot be set lower than its current value. %s < %s	A counter cannot be set lower than its current value.
0x1100a713	ADMIN	system.moab	WARN	MWM_CANNOT_LOAD_FILE	Cannot load %s file %s - %s.	Failed to load a file into Moab. Make sure it exists and that permissions are correct.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a71a	ADMIN	system.moab	WARN	MWM_FAILED_TO_WAIT_FOR_CHILD	Failed to wait for child, pid: %s, errno: %s (%s).	The wait() system call failed. Use the errno and associated message to determine possible causes.
0x1100a71d	ADMIN	system.moab	WARN	MWM_CANNOT_CHMOD_FILE	Failure changing permissions of file: '%s' to mode:'%s', errno: %s (%s).	The chmod() system call failed. Use the errno and associated message to determine possible causes.
0x1100a71e	ADMIN	system.moab	WARN	MWM_CANNOT_OPEN_FILE_WARNING	Cannot open %s file '%s', errno: %s (%s).	The fopen() system call failed. Use the errno and associated message to determine possible causes.
0x1100a722	ADMIN	system.moab	WARN	MWM_CANNOT_WRITE_FILE_WARNING	Cannot write to file '%s', errno: %s (%s).	The fwrite() system call failed. Use the errno and associated message to determine possible causes.
0x1	AD	syste	W	MWM_	Cannot close file descriptor	The close()

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a723	MIN	m.moab	ARN	CANNOT_CLOSE_FILE_DESCRIPTOR	%s, errno: %s (%s).	system call failed. Use the errno and associated message to determine possible causes.
0x1100a724	ADMIN	system.moab	WARN	MWM_CANNOT_RENAME_FILE	Failure renaming file '%s' to '%s', errno: %s (%s).	The rename() system call failed. Use the errno and associated message to determine possible causes.
0x1100a726	ADMIN	system.moab	WARN	MWM_CANNOT_BIND_TO_PORT	Cannot bind to port %s, errno: %s (%s).	The bind() system call failed. Use the errno and associated message to determine possible causes.
0x1100a72a	ADMIN	system.moab	WARN	MWM_CANNOT_SEND_TO_SOCKET	Cannot send %s byte packet, errno: %s (%s).	The send() system call failed. Use the errno and associated message to determine possible causes.
0x1100	ADMI	system.mo	WA	MWM_FAILED_	Cannot get socket %s option, errno: %s (%s).	The getsockopt() system call

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a72c	N	ab	RN	GETSOCKOPT_WARNING		failed. Use the errno and associated message to determine possible causes.
0x1100a72d	ADMIN	system.moab	WARN	MWM_FAILED_SETSOCKOPT_WARNING	Cannot set socket %s option, errno: %s (%s).	The setsockopt() system call failed. Use the errno and associated message to determine possible causes.
0x1100a738	ADMIN	system.moab	WARN	MWM_CANNOT_SET_UMASK	Failure setting umask on file '%s', errno: %s (%s).	The umask() system call failed. Use the errno and associated message to determine possible causes.
0x1100a73a	ADMIN	system.moab	WARN	MWM_FAILED_FCNTL_WARNING	Cannot set %s option on file descriptor, errno: %s (%s).	The fcntl() system call failed. Use the errno and associated message to determine possible causes.
0x1100a73	ADMIN	system.moab	WAR	MWM_CANNOT_STAT_FILE_	Cannot get stats on file '%s', errno: %s (%s).	The stat() system call failed. Use the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
b			N	WARNING		errno and associated message to determine possible causes.
0x1100a73e	ADMIN	system.moab	WARN	MWM_GET_HOSTNAME_CLIENT	Cannot get hostname of the client, errno: %s (%s).	The getnameinfo() system call failed. Use the errno and associated message to determine possible causes.
0x1100a74d	ADMIN	system.moab	WARN	MWM_CONFIG_FILE_NOT_FOUND_WARNING	Cannot locate configuration file '%s' in '%s'.	Check for the existence of this file.
0x1100a753	ADMIN	system.moab	WARN	MWM_RM_CONFIG_INVALID_VALUE	Invalid %s value '%s' specified for RM '%s'.	Check the line in the configuration file for the parameter.
0x1100a754	ADMIN	system.moab	WARN	MWM_RM_CONFIG_PROCESS_ATTR	Failed to process attribute '%s' for resource manager '%s'.	Check the line in the configuration file for the parameter.
0x1100a755	ADMIN	system.moab	WARN	MWM_RM_CONFIG_ATTR	RM attribute '%s' not handled.	Check the line in the configuration file for the parameter.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a756	ADMIN	system.moab	WARN	MWM_RM_CONFIG_TIMEOUT	Resource manager '%s' has a timeout of less than 50 ms.	Check the line in the configuration file for the parameter.
0x1100a758	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_DEFAULT_VALUE	Configuration parameter '%s[%s]' was not assigned a value. Using default.	Check the line in the configuration file to see if this behavior is desired.
0x1100a759	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_INTEGER_DEFAULT_VALUE	Configuration parameter '%s[%s]' has a value '%s' that is not an integer. Using default.	Check the line in the configuration file for the integer value.
0x1100a75a	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_DOUBLE_DEFAULT_VALUE	Configuration parameter '%s[%s]' has a value '%s' that is not a double. Using default.	Check the line in the configuration file for the double value.
0x1100a75b	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_NULL_VALUE	Configuration parameter '%s[%s]' has a NULL value.	Check the line in the configuration file.
0x1100a75c	ADMIN	system.moab	WARN	MWM_CONFIG_PARAM_INVALID	Configuration parameter '%s' has an invalid value.	Check the line in the configuration file for the attribute.
0x1	AD	syste	W	MWM_	Configuration parameter '%s[%s]' is not defined.	Check the line

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a75e	MIN	m.moab	ARN	CONFIG_PARAM_UNKNOWN		in the configuration file for the undefined parameter.
0x1100a75f	ADMIN	system.moab	WARN	MWM_CONFIG_ATTR_EXTRACTION	Configuration parameter '%s[%s]' attribute value '%s' cannot be extracted.	Check the line in the configuration file for the attribute.
0x1100a760	ADMIN	system.moab	WARN	MWM_INVALID_CONFIG_LINE	Cannot process line '%s'.	Check the line syntax against the documentation.
0x1100a763	ADMIN	system.moab	WARN	MWM_UNKNOWN_ADMINCFG_PARAMETER	Unknown ADMINCFG parameter '%s'.	Check the syntax in the configuration file.
0x1100a764	ADMIN	system.moab	WARN	MWM_UNKNOWN_MID_ATTR	Unknown identity attribute '%s'.	Check the MIDCFG lines in the configuration file.
0x1100a765	ADMIN	system.moab	WARN	MWM_UNKNOWN_AM_ATTR	Unknown account manager attribute '%s'.	Check the AMCFG lines in the configuration file.
0x1100a766	ADMIN	system.moab	WARN	MWM_UNKNOWN_ATTRIBUTE_SPECIFIED	Unknown attribute '%s' specified for %s %s.	An error occurred while parsing the configuration for the listed object. The

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						specified attribute is unknown or invalid.
0x1100a769	ADMIN	system.moab	WARN	MWM_ MONGOSERVER_ CONNECTION_FAILURE	Unable to connect to Mongo server '%s' (%s).	The program will continue to try and connect in the background.
0x1100a76e	ADMIN	system.moab	WARN	MWM_ EVENT_ QUERY_ ODBC	Event querying is only supported with ODBC.	Check the USEDATABASE option.
0x1100a770	ADMIN	system.moab	WARN	MWM_DB_ CONNECT	Cannot connect to DB--falling back to file and memory-based storage (%s).	Verify that the database is running.
0x1100a772	ADMIN	system.moab	WARN	MWM_ DATABASE_ STATS	Unable to retrieve statistics from the database.	Verify that the database is running.
0x1100a774	ADMIN	system.moab	WARN	MWM_ SERVER_ CONNECTION_FAILED_ TRYING_ FALLBACK	The system was unable to connect to the server %s:%s - attempting fallback server %s.	Make sure the server's address is correct and it is running.
0x1100a775	ADMIN	system.moab	WARN	MWM_ PRIMARY_ SERVER_ FAILED_ TRYING_ BACKUP	The system was unable to connect to the server %s (%s:%s) - trying backup server (%s:%s).	Make sure the server's address is correct and it is running.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a778	ADMIN	system.moab	WARN	MWM_INVALID_REDUCE_CLIENTMAXCONNECTIONS	Reducing CLIENTMAXCONNECTIONS to %s from %s not allowed during runtime.	Decreasing the value of CLIENTMAXCONNECTIONS cannot be done during runtime.
0x1100a785	ADMIN	system.moab	WARN	MWM_UNABLE_TO_START_JOB	Cannot start job %s. (%s)	The job failed to start.
0x1100a788	ADMIN	system.moab	WARN	MWM_UNABLE_TO_ALLOCATE_NODES_FOR_JOB	Cannot allocate nodes for job %s. (%s)	Cannot allocate a node list that matches the requirements for this job. This might not be serious since multiple passes may occur.
0x1100a7d2	ADMIN	system.moab	WARN	MWM_UNABLE_LOAD_PBS_JOB	Cannot load PBS job '%s'.	Could not load a job discovered from a PBS resource manager into Moab.
0x1100a7e3	ADMIN	system.moab	WARN	MWM_TRUNCATING_ATTRIBUTE_FOR_CLASS	Truncating %s for class: %s (rm reports: %s; Moab enforces: %s).	The resource manager reports a certain value for a class, but Moab has been instructed to keep it within certain limits.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						The value will be truncated to keep it within the limits.
0x1100a7e4	ADMIN	system.moab	WARN	MWM_UNEXPECTED_JOB_STATE	Unexpected job state '%s' detected for job %s.	The listed job was found to be in a state that was not expected. This may or may not be an error condition.
0x1100a821	ADMIN	system.moab	WARN	MWM_CLOCK_SKEW_DETECTED	Clock skew detected (%s time for job %s in %s).	A reported time associated with the job appears to be wrong. This could be because of a lack of synchronization between system clocks on all nodes.
0x1100a83e	ADMIN	system.moab	WARN	MWM_INVALID_WIKI_ATTRIBUTE	Encountered invalid wiki attribute while reading '%s'.	Check the syntax of the attribute.
0x1100a83f	ADMIN	system.moab	WARN	MWM_DUPLICATE_WIKI_ATTRIBUTE	Wiki attribute '%s' is already set.	Check for duplicate instances of the attribute.
0x1100a840	ADMIN	system.moab	WARN	MWM_VM_UNSUPPORTED_WIKI_ATTRIBUTE	Wiki attribute '%s' is unsupported for VM creation.	Remove the attribute.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a843	ADMIN	system.moab	WARN	MWM_ADD_NODE_FAILURE	Cannot add node '%s' to global node table. Index is already used.	Cannot have two nodes with the same name.
0x1100a844	ADMIN	system.moab	WARN	MWM_HT_ADD_NODE_FAILURE	Cannot add node '%s' to hash table. Index is already used.	Cannot have two nodes with the same name.
0x1100a846	ADMIN	system.moab	WARN	MWM_VM_MIGRATION_FAILURE	Cannot migrate VM '%s'.	The VM might not be eligible for migration.
0x1100a84c	ADMIN	system.moab	WARN	MWM_UNEXPECTED_SUBCOMMAND_RECEIVED	Unexpected subcommand '%s' received.	The communication from a Moab client includes an unknown subcommand.
0x1100a84d	ADMIN	system.moab	WARN	MWM_UNABLE_TO_REGISTER_JOB_AM	Unable to register job %s with accounting manager for job %s. reason: '%s' message:'%s'.	The accounting manager was unable to register the listed job for a certain action. An optional reason and/or message might be given to assist in diagnosis.
0x1100a851	ADMIN	system.moab	WARN	MWM_IMPROPER_VM_MIGRATION_DECISION	The migration decision for the VM was not properly set up.	The information indicating the destination node is missing.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a85b	ADMIN	system.moab	WARN	MWM_JOB_UPDATE_NULL_STARTTIME	Start time is NULL for job update.	Specify a start time that is greater than zero.
0x1100a85c	ADMIN	system.moab	WARN	MWM_JOB_UPDATE_NULL_DISPATCHTIME	Dispatch time is NULL for job update.	Specify a dispatch time that is greater than zero.
0x1100a862	ADMIN	system.moab	WARN	MWM_UNABLE_TO_REGISTER_RESERVATION_AM	Unable to register reservation %s with accounting manager for %s processors for reservation %s.	The accounting manager was unable to register the listed reservation for a certain action.
0x1100a863	ADMIN	system.moab	WARN	MWM_DEPRECATED_PARAMETER_VALUE	Deprecated value '%s' specified for parameter '%s'. %s	The listed value is no longer valid for this parameter. A hint might be provided with the message. Check the most recent documentation for the software version.
0x1100a864	ADMIN	system.moab	WARN	MWM_INVALID_FILE_ATTRIBUTES_WARNING	Invalid value '%s' specified for %s (%s).	Checking a file to see whether it exists, is executable, etc, has produced unexpected results.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a865	ADMIN	system.moab	WARN	MWM_CANNOT_SET_JOB_ATTRIBUTE_VIA_TEMPLATE_WARNING	Cannot set %s %s via template %s.	Failed to set the listed attribute to the listed value for a specified job template.
0x1100a866	ADMIN	system.moab	WARN	MWM_KILL_PROCESS_FAILURE	Unable to kill process %s.	The system tried to kill the given process and failed.
0x1100a86a	ADMIN	system.moab	WARN	MWM_NODELIST_STRING_BUFFER	Insufficient buffer space to convert a node list into a string.	The buffer must be larger to hold all the nodes.
0x1100a86b	ADMIN	system.moab	WARN	MWM_MAX_NODES_EXCEEDED	The maximum number of nodes associated with a reservation has been exceeded.	The number of nodes must be reduced.
0x1100a871	ADMIN	system.moab	WARN	MWM_VC_ALREADY_ADDED	The virtual container '%s' is already an ancestor of VC '%s'.	Cannot create a circular chain, must maintain a hierarchical structure.
0x1100a872	ADMIN	system.moab	WARN	MWM_VC_REMOVAL_FAILURE	The virtual container '%s' cannot be removed.	This is an internal error.
0x1100a874	ADMIN	system.moab	WARN	MWM_RESERVATION_JOB_NOT_FOUND	Unable to find the job for reservation '%s'.	The host job for the reservation is NULL.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a875	ADMIN	system.moab	WARN	MWM_SINGLE_USE_RESERVATION_DESTRUCTION	Unable to destroy a single-use reservation.	This is an internal error.
0x1100a878	ADMIN	system.moab	WARN	MWM_UNEXPECTED_JOB_SUBMISSION_POLICY	The system encountered an unexpected job submission policy (%s).	The job submission policy did not match a defined policy.
0x1100a879	ADMIN	system.moab	WARN	MWM_SIMULATION_JOB_RECORDS	Unable to simulate workload by creating job records (1000 attempts).	The system might be low on memory.
0x1100a880	ADMIN	system.moab	WARN	MWM_JOB_TRANSITION_XML_MESSAGE	Unable to add messages to job '%s' transition XML.	The system might be low on memory.
0x1100a884	ADMIN	system.moab	WARN	MWM_PBS_API_STALE	PBS API is stale - re-initializing.	Re-initializing the PBS environment.
0x1100a885	ADMIN	system.moab	WARN	MWM_UNABLE_TO_GET_PBS_QUEUE_INFO	Cannot process PBS queue info for RM %s (node %s) - no data available.	Unable to get any information on the PBS queues. Make sure that there was at least a queue set up in PBS.
0x1	AD	syste	W	MWM_	Cannot set job '%s' attribute	There was a

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a886	MIN	m.moab	ARN	UNABLE_TO_SET_JOB_ATTRIBUTE	'%s:%s' to '%s' (rc: %s; '%s').	problem while changing the job attribute and the error status was displayed in rc.
0x1100a887	ADMIN	system.moab	WARN	MWM_UNABLE_TO_CONNECT_PBS_SCHEDULER	Cannot connect to PBS event/scheduler port %s.	Ensure the PBS scheduler is running and listening on the specified port.
0x1100a888	ADMIN	system.moab	WARN	MWM_NODE_UNUSABLE_NO_DISK	Idle node %s is unusable (inadequate disk space in /var).	Ensure that the node has sufficient disk space.
0x1100a889	ADMIN	system.moab	WARN	MWM_NODE_UNUSABLE_BAD_STATE	Node '%s' is unusable in state 'NONE'.	The node has become unusable because of its state being NONE.
0x1100a88a	ADMIN	system.moab	WARN	MWM_UNABLE_TO_FIND_USERS_GID	Cannot locate OS GID information for user '%s' - ignoring user.	Moab was unable to find the GID of this user. Make sure that this user has a GID.
0x1100a88b	ADMIN	system.moab	WARN	MWM_UNABLE_TO_FIND_USERS_UID	Cannot locate OS information for user '%s' - ignoring user - %s.	Moab was unable to find the user on the system. Make sure that this user exists.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a88c	ADMIN	system.moab	WARN	MWM_UNABLE_TO_FIND_GID_LIST	Cannot locate OS group list information for user '%s' - ignoring user.	Moab was unable to find the group list for this user.
0x1100a88d	ADMIN	system.moab	WARN	MWM_TIMEOUT	Command '%s' timed out, or wait failed after %s seconds.	Increasing the TIMEOUT settings in moab.cfg might help.
0x1100a88e	ADMIN	system.moab	WARN	MWM_INSUFFICIENT_POLICIES	Insufficient policies specified; hpolicy=%s,spolicy=%s.	Please revise your policies along with their actions.
0x1100a88f	ADMIN	system.moab	WARN	MWM_NO_STDOUT	Request succeeded with no stdout but stderr='%s'.	Typically there will also be stdout when there is stderr. Depending on the request this might be the intended result.
0x1100a890	ADMIN	system.moab	WARN	MWM_CANNOT_CONNECT_WIKI	Cannot connect to Wiki event port %s.	Failure to connect to the Wiki event port.
0x1100a891	ADMIN	system.moab	WARN	MWM_MISSING_COLON_STR	Colon delimiter not located in %s wiki string '%s...' in %s.	Check that the string contains the right format.
0x1100a892	ADMIN	system.moab	WARN	MWM_ADD_DEPENDENCY_FAIL	Failed to add dependencies to job %s's submission.	There was a problem in adding the dependencies to the job.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a893	ADMIN	system.moab	WARN	MWM_INVALID_PRIORITY_FUNCTION	Invalid priority function '%s' on job '%s'.	The priority function applied to the job was invalid.
0x1100a894	ADMIN	system.moab	WARN	MWM_MISSING_JOB_REQ	Invalid job '%s'; no requirements.	The job was invalid because it was missing the requirements.
0x1100a895	ADMIN	system.moab	WARN	MWM_MISSING_JOB_REQ_AT_INDEX	Invalid job %s; no requirement at index %s.	The job was invalid because an index was missing requirements.
0x1100a896	ADMIN	system.moab	WARN	MWM_INVALID_WIKI_STR_MISSING_EQUAL	Malformed wiki string '%s' - no '='.	The wiki string was missing an equal sign '='.
0x1100a897	ADMIN	system.moab	WARN	MWM_INVALID_EMPTY_WIKI_STR	Malformed wiki string '%s' - EOF.	The wiki string was empty.
0x1100a898	ADMIN	system.moab	WARN	MWM_INCORRECT_STAGE_LOC	stage-data source location is being incorrectly reported via wiki '%s' != '%s'.	The stage data source location was incorrectly reported in wiki.
0x1100a899	ADMIN	system.moab	WARN	MWM_INVALID_VM_OBJECT_ID	VM '%s' is not a valid object, ignoring.	The VM does not have a valid object ID.
0x1	AD	syste	W	MWM_	Could not parse JSON	The JSON

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a89a	MIN	m.moab	ARN	INVALID_JSON_CLUSTER	cluster query data from MWS RM (%s): %s.	construct(s) for the cluster might contain some invalid syntax.
0x1100a89b	ADMIN	system.moab	WARN	MWM_INVALID_JSON_WORKLOAD	Could not parse JSON workload query data from MWS RM (%s): %s.	The JSON construct(s) for the workload might contain some invalid syntax.
0x1100a89c	ADMIN	system.moab	WARN	MWM_MISSING_REQ_PROPERTIES	JSON cluster query data from MWS RM (%s) does not contain required properties (%s, %s, %s).	The JSON constructs for the cluster query data are missing the required properties.
0x1100a89d	ADMIN	system.moab	WARN	MWM_INVALID_JSON_CLUSTER_OBJECT	JSON cluster query data from MWS RM (%s) is not a valid object.	Review the JSON construct for the cluster query data to ensure its syntax is correct.
0x1100a89e	ADMIN	system.moab	WARN	MWM_INVALID_JSON_WORKLOAD_OBJECT	JSON workload query data from MWS RM (%s) is not a valid object.	Review the JSON construct for the workload query data to ensure its syntax is correct.
0x1100	ADMI	system.mo	WA	MWM_EMPTY_	Empty %s response from RM (%s).	The response from the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a89f	N	ab	RN	RESPONSE		resource manager query was empty.
0x1100a8a0	ADMIN	system.moab	WARN	MWM_INVALID_NODE_DATA	Nodes data from MWS RM (%s) is not a valid object.	The response from the Moab Web Services resource manager query was empty.
0x1100a8a1	ADMIN	system.moab	WARN	MWM_CREATE_RESERVATION_FAIL	Cannot create requested reservation (%s).	The request to create the given reservation has failed.
0x1100a8a2	ADMIN	system.moab	WARN	MWM_LOCATE_RSVPROFILE_FAIL	Cannot locate RSVPROFILE '%s'.	Moab failed to find the given RSVPROFILE. Confirm that the file exists.
0x1100a8a3	ADMIN	system.moab	WARN	MWM_LOCATE_RSV_PARENT_FAIL	Cannot locate parent '%s' for reservation '%s'.	Moab failed to locate the parent of the given reservation.
0x1100a8a4	ADMIN	system.moab	WARN	MWM_LOCATE_COMMAND_FAIL	Cannot locate command '%s'.	Moab failed to locate the given command. Confirm that the command exists.
0x1100a8a5	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_SCHED_CMD	Received unexpected sched command '%s'.	Received an unexpected mschedctl

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						command. Confirm that the used option is supported.
0x1100a8a6	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_EVENT	Unsupported event '%s' from RM '%s'.	The given event is not supported by the given resource manager.
0x1100a8a7	ADMIN	system.moab	WARN	MWM_VM_MIGRATE_FAIL	VM %s should migrate from node %s but cannot locate valid destination - %s (policy).	Attempt to migrate the given VM from the given node failed. Check that the destination is valid.
0x1100a8a8	ADMIN	system.moab	WARN	MWM_JOB_START_FAIL	Start of system job %s failed; no action specified.	Failed to start a job because there was no action specified.
0x1100a8a9	ADMIN	system.moab	WARN	MWM_VMTRACKING_JOB_FAIL	VM '%s' reported a system job failure on VMTracking job '%s'.	The given VM reported it failed on the given VMTracking job.
0x1100a8aa	ADMIN	system.moab	WARN	MWM_VMTRACKING_EXCEED_WALTIME	VM '%s' exceeded its allocated walltime. VMTracking job '%s' (pointing to job '%s').	The given VM has exceeded its allocated walltime on the associated VM tracking

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						job.
0x1100a8ac	ADMIN	system.moab	WARN	MWM_UNKNOWN_POWER_STATE	No RM can report node '%s' power state for system job '%s'.	No resource manager can report the power state for the given nodes on the given job.
0x1100a8ad	ADMIN	system.moab	WARN	MWM_ADD_GLOBAL_NODE_FAIL	Cannot add global node '%s'.	Failed to add the given global node.
0x1100a8ae	ADMIN	system.moab	WARN	MWM_UNKNOWN_CLIENT	Client ID '%s' is unknown.	Moab failed to recognize the name/ID of the given client.
0x1100a8af	ADMIN	system.moab	WARN	MWM_JOB_DEBIT_ACCOUNT	Unable to charge funds for job.	The account manager failed to debit the account for the job.
0x1100a8b0	ADMIN	system.moab	WARN	MWM_JOB_RESERVE_ACCOUNT	Unable to reserve funds for job (Reason: %s).	The account manager failed to reserve funds on the account for the job.
0x1100a8b1	ADMIN	system.moab	WARN	MWM_CANCEL_LEIN	Unable to cancel lien for instance '%s' (Reason: %s).	The account manager failed to release the lien.
0x1100	ADMI	system.mo	WA	MWM_RESERVATI	Unable to reserve funds for reservation (Reason: %s).	The account manager failed

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a8b2	N	ab	RN	ON_RESERVE_ACCOUNT		to reserve funds on the account for the reservation.
0x1100a8b3	ADMIN	system.moab	WARN	MWM_JOB_TASK_DISTRIBUTION	The system cannot distribute the tasks allocated for a job.	Check the tasks specified in the job.
0x1100a8b4	ADMIN	system.moab	WARN	MWM_JOB_DEFAULT_CLASS	Job cannot run with default class '%s'.	Check the limits set on the class.
0x1100a8b5	ADMIN	system.moab	WARN	MWM_RM_START_JOB	Cannot start job through a resource manager.	The resource manager might not be set to run the job.
0x1100a8b6	ADMIN	system.moab	WARN	MWM_JOB_DEPENDENCY_UPDATE	Cannot find job '%s' to update dependency '%s' for job '%s'.	The dependency job for the specified job is missing.
0x1100a8b7	ADMIN	system.moab	WARN	MWM_EXPIRED_CHECKPOINT	The checkpoint has expired.	Items within the checkpoint might no longer be valid.
0x1100a8b8	ADMIN	system.moab	WARN	MWM_BAD_CHECKPOINT_LINE	The system encountered an incorrectly formed checkpoint line for key '%s'.	All lines must end with a NEWLINE character.
0x1100a8b9	ADMIN	system.moab	WARN	MWM_CONVERT_XML_FROM_STRING	XML data cannot be obtained from an XML string ('%s').	There was an error converting from a string

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						that should contain XML into internal XML data structures.
0x1100a8ba	ADMIN	system.moab	WARN	MWM_CONVERT_XML_TO_STRING	An XML string cannot be constructed from XML data.	There was an error converting from internal XML data structures into an XML string representation.
0x1100a8bb	ADMIN	system.moab	WARN	MWM_SOCKET_OPERATION	Cannot %s message on sd %s within %s second timeout.	There is a communication error with sockets.
0x1100a8bc	ADMIN	system.moab	WARN	MWM_COMPLETED_JOB_RECORD	Could not create job record for completed job %s - %s.	The system might be low on memory.
0x1100a8bd	ADMIN	system.moab	WARN	MWM_CREATE_TEMPLATE_JOB_DEPENDENCY	Could not create template job dependency %s - %s.	The system might be low on memory.
0x1100a8be	ADMIN	system.moab	WARN	MWM_CANNOT_FIND_SMP_NODE_BY_FEATURE	Could not find SMP node by feature '%s'.	The feature did not match any of the SMP nodes.
0x1100a8bf	ADMIN	system.moab	WA	MWM_CHECKPOIN	Could not process completed job from checkpoint.	Examine the checkpoint

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
			RN	T_PROCESS_COMPLETED_JOB		entry for the job.
0x1100a8c0	ADMIN	system.moab	WARN	MWM_CANNOT_FIND_SMP_NODE_IN_PARTITION	Could not find SMP node in partition '%s'.	The feature did not match any of the SMP nodes in the partition.
0x1100a8c1	ADMIN	system.moab	WARN	MWM_JOB_REPORTED_BY_RM_NOT_OWNER	Job '%s' is being reported by RM '%s' but is owned by RM '%s'.	The resource manager reporting does not own the job.
0x1100a8c2	ADMIN	system.moab	WARN	MWM_PEER_RM_UNKNOWN_LANGUAGE	Peer RM '%s' reported unknown language: '%s'.	The language does not match a known format.
0x1100a8c3	ADMIN	system.moab	WARN	MWM_PEER_RM_UNKNOWN_SUBLANGUAGE	Peer RM '%s' reported unknown sub-language: '%s'.	The language does not match a known format.
0x1100a8c5	ADMIN	system.moab	WARN	MWM_JOB_ARCH_VALUE	Job '%s' does not have a valid arch (architecture) value '%s'.	Check the specified value for the architecture.
0x1100a8c6	ADMIN	system.moab	WARN	MWM_INVALID_HOST_REQ	Job '%s' does not have a valid host requirement '%s'.	Check the specified value for the requirement.
0x1100a8c	ADMIN	system.moab	WAR	MWM_JOB_OPSYS_VALUE	Job '%s' does not have a valid operating system value '%s'.	Check the specified value for the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
7			N			operating system.
0x1100a8c8	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_REQ	Resource requirement '%s' not supported.	The requirement specified is unsupported.
0x1100a8c9	ADMIN	system.moab	WARN	MWM_NO_TASKS	Job loaded in active state with no tasks allocated.	Jobs must have at least one task.
0x1100a8ca	ADMIN	system.moab	WARN	MWM_SINGLE_ITERATION_JOB_COMPLETION	Scheduler cannot handle job completion in a single iteration.	The job must not start and complete while the scheduler is sleeping.
0x1100a8cb	ADMIN	system.moab	WARN	MWM_MAX_TASKS_EXCEEDED	The number of tasks associated with a job has exceeded the maximum (%s).	The number of tasks must be reduced or the scheduler must be configured to accept more tasks.
0x1100a8cc	ADMIN	system.moab	WARN	MWM_NODE_OUT_OF_RANGE	Job '%s' node index (%s) at task list index (%s) is out of range.	This is an internal limit.
0x1100a8cd	ADMIN	system.moab	WARN	MWM_NODE_NULL	Job '%s' node index (%s) at task list index (%s) is NULL.	This is an internal limit.
0x1100	ADMI	system.mo	WA	MWM_NODESET_	Nodeset constraints prevent use of task for job '%s':%s	The specified nodeset cannot

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a8ce	N	ab	RN	CONSTRAINTS	at %s.	run the task.
0x1100a8cf	ADMIN	system.moab	WARN	MWM_DEFAULT_WALLTIME	Job assigned default walltime limit (%s).	Unlimited or no walltime limit specified.
0x1100a8d0	ADMIN	system.moab	WARN	MWM_PARTITION_ACCESS	Job cannot access requested partitions (%s).	The partition access list disallows the job.
0x1100a8d1	ADMIN	system.moab	WARN	MWM_UNABLE_TO_ALLOCATE_TASKS_FOR_JOB	Cannot allocate tasks for job at %s.	The system might be low on memory.
0x1100a8d2	ADMIN	system.moab	WARN	MWM_IGNORING_PARTIAL_RANGE	Ignoring partial time range since full range previously located.	The system will use the full range instead.
0x1100a8d3	ADMIN	system.moab	WARN	MWM_DESTINATION_RM	Cannot locate a valid destination resource manager for job.	The submitted job could not be sent to a resource manager.
0x1100a8d4	ADMIN	system.moab	WARN	MWM_JOB_CREDENTIALS	Cannot authenticate the submitted job (Reason: %s).	The user for the job is not a member of a group or account with access.
0x1100	ADMI	system.mo	WA	MWM_SMPNODE_	Could not find SMPNode by feature %s (%s).	None of the nodes has the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a8d5	N	ab	RN	BY_ FEATURE		feature specified.
0x1100a8d6	ADMIN	system.moab	WARN	MWM_ SMPNODE_ BY_ PARTITION	Could not find SMPNode in partition %s.	The SMPNode specified is not in the given partition.
0x1100a8d7	ADMIN	system.moab	WARN	MWM_RSV_ ATTR_TO_ STRING	Reservation '%s' attribute '%s' could not be converted to a string.	There is no string conversion routine for that attribute type.
0x1100a8d8	ADMIN	system.moab	WARN	MWM_AM_ STATUS	Account manager sent failure message - %s.	Check status message.
0x1100a8d9	ADMIN	system.moab	WARN	MWM_AM_ FAILURE	Native accounting manager call '%s' failed using input XML '%s'.	Check XML syntax.
0x1100a8da	ADMIN	system.moab	WARN	MWM_AM_ INSUFFICIENT_FUNDS	Account manager - Insufficient funds '%s'.	Validate that the user has access to account.
0x1100a8db	ADMIN	system.moab	WARN	MWM_ MIGRATE_ JOB	Unable to migrate job '%s' to RM '%s' (%s).	Check the error message.
0x1100a8dc	ADMIN	system.moab	WARN	MWM_ RESERVE_ PRIORITY_ JOB	Unable to reserve priority job.	Check the error message.
0x1	AD	syste	W	MWM_SYNC_	Job '%s' not synchronized to	The two jobs

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a8dd	MIN	m.moab	ARN	JOB	start with job '%s'.	must start at the same time.
0x1100a8de	ADMIN	system.moab	WARN	MWM_SYNC_JOB_REQUEUE	Job '%s' could not start. Requeuing any synchronized jobs.	The other jobs should be back on the queue.
0x1100a8df	ADMIN	system.moab	WARN	MWM_UNHANDLED_PLUGIN_EXCEPTION	A node allocation plugin '%s' encountered an unhandled exception '%s'.	Consult the documentation for the plug-in.
0x1100a8e0	ADMIN	system.moab	WARN	MWM_PLUGIN_LOADED_FAILURE	Error loading node allocation plugin '%s' for partition '%s' %s.	A NodeAllocation plug-in was not loaded because of an error. Default node allocation will be used.
0x1100a8e2	ADMIN	system.moab	WARN	MWM_UNKNOWN_JOB_DEPENDENCY	Unknown job dependency '%s' on job.	The job is trying to use a dependency that is unknown.
0x1100a8e3	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_JOB_DEPENDENCY	Unknown job dependency type '%s' on job.	The job is trying to use a dependency type that is unsupported.
0x1100a8e4	ADMIN	system.moab	WARN	MWM_MISSING_JOB_DEPENDENCY	Cannot find dependency job. MissingDependencyAction is '%s'.	Check for the existence of the job dependency.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a8e5	ADMIN	system.moab	WARN	MWM_PARTITION_REP_NODE	Corrupt partition representative node.	Check the representative node for the partition.
0x1100a8e6	ADMIN	system.moab	WARN	MWM_PARTITION_ATTRIBUTE	Partition attribute '%s' is not configurable.	Consult the documentation to see which attributes can be configured.
0x1100a8e7	ADMIN	system.moab	WARN	MWM_UNABLE_TO_LOCATE_NODE	Unable to locate specified nodes for job.	Could not find a node in the job's node list.
0x1100a8e8	ADMIN	system.moab	WARN	MWM_NODES_MISSING_FROM_FEASIBLE_LIST	Specified node(s) not found in feasible hostlist.	Could not find a node.
0x1100a8e9	ADMIN	system.moab	WARN	MWM_HOSTLIST_HAS_TOO_FEW_TASKS	A hostlist has too few tasks available for job '%s':'%s' (%s < %s).	More nodes are needed to satisfy the task requirements.
0x1100a8ea	ADMIN	system.moab	WARN	MWM_TASKS_REMAINING	A hostlist was unable to handle all tasks (%s remain).	More nodes are needed to satisfy the task requirements.
0x1100a8eb	ADMIN	system.moab	WARN	MWM_NODE_DOWN	Unable to detect node '%s' for '%s' seconds. Marking it down or removing it.	Make sure the node is up and running.
0x1100	ADMI	system.mo	WA	MWM_NODE_	Unable to reset node. Node list empty.	Must specify a valid node to

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a8ec	N	ab	RN	RESET_ EMPTY		reset.
0x1100a8ed	ADMIN	system.moab	WARN	MWM_ NODE_ RESET_URL	Unable to reset node. NODEPOWERURL not specified.	Must specify a valid URL for the node to reset.
0x1100a8ee	ADMIN	system.moab	WARN	MWM_AM_ DOWN	The account manager is not currently running.	Check the status of the account manager.
0x1100a8ef	ADMIN	system.moab	WARN	MWM_ KEYBOARD_ ACTIVITY_ PREEMPT_ JOB	Keyboard activity on node prevented job preemption.	Jobs can be preempted only if the keyboard is idle.
0x1100a8f0	ADMIN	system.moab	WARN	MWM_ KEYBOARD_ ACTIVITY_ SET_NODE_ STATE	Keyboard activity on node prevented setting the node state to '%s'.	Node states can be changed only if the keyboard is idle.
0x1100a8f1	ADMIN	system.moab	WARN	MWM_JOB_ MISMATCHED_TIMES	Fixing job '%s' with invalid '%s' times (%s - %s).	Check the times for the specified job.
0x1100a8f2	ADMIN	system.moab	WARN	MWM_JOB_ OPSYS	Cannot add operating system '%s' to job.	Check the type of operating system specified.
0x1100a8f3	ADMIN	system.moab	WARN	MWM_JOB_ ARCH	Cannot add architecture '%s' to job.	Check the type of architecture specified.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a8f4	ADMIN	system.moab	WARN	MWM_LOCATE_AM	Cannot locate the account manager '%s'.	Check the account manager command option syntax.
0x1100a8f5	ADMIN	system.moab	WARN	MWM_LOCATE_RM	Cannot locate the resource manager '%s'.	Check the resource manager command option syntax.
0x1100a8f6	ADMIN	system.moab	WARN	MWM_LOCATE_RMID	Cannot locate the resource manager ID '%s'.	Check the ID command option syntax.
0x1100a8f7	ADMIN	system.moab	WARN	MWM_LOCATE_PARTITION	Cannot locate the partition '%s'.	Check the partition command option syntax.
0x1100a8f8	ADMIN	system.moab	WARN	MWM_RM_QUEUE_MODIFY	Command to modify RM queue failed on resource manager %s - '%s'.	Queue may be configured to reject modify requests.
0x1100a8f9	ADMIN	system.moab	WARN	MWM_RM_QUEUE_CREATE	Command to create RM queue failed on resource manager %s - '%s'.	System may be configured to reject queue creation requests.
0x1100a8fa	ADMIN	system.moab	WARN	MWM_RM_QUEUE_CREATE_MISSING_ARGS	Command to create RM queue failed - arguments missing.	The user must supply the needed arguments to the command.
0x1	AD	syste	W	MWM_	An attempt was made to	Static resource

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a8fb	MIN	m.moab	ARN	STATIC_RM_DESTRUCTION	destroy a static resource manager.	managers cannot be destroyed.
0x1100a8fc	ADMIN	system.moab	WARN	MWM_ADD_SYSTEM_USER	Unable to create a new user '%s' in the system.	The system might be low on memory.
0x1100a8fd	ADMIN	system.moab	WARN	MWM_ADD_PARTITION	The system was unable to create partition '%s'.	The system might be low on memory.
0x1100a8fe	ADMIN	system.moab	WARN	MWM_CORE_LIMIT	System core limit set to %s (complete core files might not be generated).	Expand the system core limit to ensure the complete core dump can be saved.
0x1100a8ff	ADMIN	system.moab	WARN	MWM_KEY_FILE_PERMISSIONS	The .moab.key file exists, but the file permissions prevent access (%s).	Check the ownership permissions on the file.
0x1100a900	ADMIN	system.moab	WARN	MWM_STATS_PERIOD_TYPE	The system could not process stats for period type %s.	'Day' is the only period type currently supported.
0x1100a901	ADMIN	system.moab	WARN	MWM_STATS_BUFFER_SIZE	The system could not process stats for period type %s (buffer too small).	The buffer allocated was too small to hold the data.
0x1100a902	ADMIN	system.moab	WARN	MWM_STATS_FILE	The system could not create the stats file '%s'.	Check the path and user permissions on the directory.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a903	ADMIN	system.moab	WARN	MWM_JOB_NO_ACCOUNT	No account specified for job '%s'.	Check the job for an account specification.
0x1100a904	ADMIN	system.moab	WARN	MWM_SET_JOBATTR_FAIL	Cannot set attribute '%s' to value '%s' on jobmatch '%s'.	Failed to set the given attribute to the given value on the given job.
0x1100a905	ADMIN	system.moab	WARN	MWM_JOBATTR_NOT_SUPPORTED	JobAttr not supported. '%s'.	The given attribute is not a supported job attribute.
0x1100a906	ADMIN	system.moab	WARN	MWM_INVALID_TRIGGER_DEFINITION	Invalid trigger definition: %s.	The given trigger is invalid. Check that the given trigger has been defined.
0x1100a907	ADMIN	system.moab	WARN	MWM_ATTRIBUTE_NOT_HANDLED	System attribute '%s' not handled.	Check that the given attribute was spelled correctly.
0x1100a908	ADMIN	system.moab	WARN	MWM_QOS_IN_PARAM_NOT_FOUND	Cannot locate QOS '%s' for parameter %s.	Make sure that the given QOS exists.
0x1100a909	ADMIN	system.moab	WARN	MWM_INVALID_PROFILEDURATION_VAL	Invalid PROFILEDURATION specified, modified internally to %s (see documentation).	The entered PROFILEDURATION value is invalid. Moab uses the given value instead.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a90a	ADMIN	system.moab	WARN	MWM_NO_DATA_STAGING_PATH	No path in data staging specification '%s' (bad format).	Verify the data staging path is specified.
0x1100a90b	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_RM_DATA_STAGING	Cannot stage-out stdout/stderr (unsupported RM type '%s').	Failed to stage-out stdout/stderr because the given resource manager does not support such feature.
0x1100a90c	ADMIN	system.moab	WARN	MWM_RM_DATA_ON_NON_EXISTING_JOB	Storage RM '%s' reporting data operation for non-existent job '%s'.	The given resource manager is reporting data operation on the non-existing job.
0x1100a90d	ADMIN	system.moab	WARN	MWM_INVALID_RM_DATA_STAGE	Data stage for RM '%s' not possible as it has no nodelist.	Check CLUSTERQUERYURL to ensure it at least has a nodelist.
0x1100a90e	ADMIN	system.moab	WARN	MWM_DATA_STAGE_IN_FAIL	Data stage in failed for job '%s' file '%s' (%s).	Failed to complete the data staging in operation for the job on the given file due to error(s).
0x1100a90f	ADMIN	system.moab	WARN	MWM_UNABLE_TO_REMOVE_	Cannot remove data staging block for job '%s'.	Failed to remove the data staging block for the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
				DATA_ STAGE		given job.
0x1100a910	ADMIN	system.moab	WARN	MWM_ DATA_ STAGE_OUT_ FAIL	Data stage out failed for job '%s' file '%s' (%s).	Failed to complete the data staging out operation for the job on the given file due to error (s).
0x1100a911	ADMIN	system.moab	WARN	MWM_JOB_ INVALID_ OPSYS	Job '%s' cannot request OS '%s').	The requested operating system is not available for the job.
0x1100a912	ADMIN	system.moab	WARN	MWM_ NODE_ BUFFER_ OVERFLOW	Node buffer is full (check license and MAXNODE parameter).	Try increasing the node buffer size (MAXNODE).
0x1100a913	ADMIN	system.moab	WARN	MWM_ REMOVE_ NODE_ WITH_ RESERVATION	Unable to remove node '%s' because of reservation references.	Remove the reservations from the node.
0x1100a914	ADMIN	system.moab	WARN	MWM_JOB_ PURGE_RM_ INACTIVE	Unable to purge job '%s' because the resource manager '%s' is inactive.	Check the status of the resource manager.
0x1100a915	ADMIN	system.moab	WARN	MWM_JOB_ PURGE_RM_ NO_ RESOURCES	Unable to purge job '%s' because the resource manager '%s' is reporting no resources.	Check the status of the resource manager.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a916	ADMIN	system.moab	WARN	MWM_JOB_NOT_DETECTED	Job '%s' in state '%s' no longer detected (Last Detected %s > PurgeTime %s).	The job might have been purged in the meantime.
0x1100a917	ADMIN	system.moab	WARN	MWM_BACKFILL_DEPTH_REACHED	The backfill depth (BFDEPTH) has been reached so no more jobs will be backfilled this iteration.	Wait for the next iteration or increase the depth.
0x1100a91c	ADMIN	system.moab	WARN	MWM_CHECKPOINT_CREATE_RSV_FROM_XML	Unable to create a reservation from checkpoint XML.	The system might be low on memory.
0x1100a91d	ADMIN	system.moab	WARN	MWM_JOB_CACHE_REMOVAL	Failed to remove job %s (ID = %s) from the cache.	The job was missing from the system hash table.
0x1100a91e	ADMIN	system.moab	WARN	MWM_INVALID_CRED_VALUE	Invalid credential value '%s'.	Check the syntax in the configuration file.
0x1100a91f	ADMIN	system.moab	WARN	MWM_INVALID_CRED_ATTR	Invalid credential attribute '%s'.	Check the syntax in the configuration file.
0x1100a920	ADMIN	system.moab	WARN	MWM_MAX_JOBS_EXCEEDED	The maximum number of jobs has been exceeded.	Increase the value of the MAXJOB setting.
0x1100a92	ADMIN	system.moab	WAR	MWM_JOB_FAILED_PROCESSIN	A job failed while processing PBS resources.	May not have been able to locate host or

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
1			N	G_PBS		vnode.
0x1100a923	ADMIN	system.moab	WARN	MWM_AM_INSUFFICIENT_BALANCE	Insufficient balance in primary account '%s' to run job '%s' (attempting fallback credentials).	Validate that the user has access to account.
0x1100a924	ADMIN	system.moab	WARN	MWM_AM_JOB_SUBMIT_VALIDATION	Job submission validation failed for job '%s' -- taking action '%s'.	Validate that the job has access.
0x1100a925	ADMIN	system.moab	WARN	MWM_TOO_MANY_NODE_SETS	The maximum number of node sets has been exceeded.	This is a configurable setting.
0x1100a926	ADMIN	system.moab	WARN	MWM_CLASS_SET_LIST_INVALID	The specified class set list is invalid '%s'.	Check the documentation for valid classes.
0x1100a928	ADMIN	system.moab	WARN	MWM_RM_START	Cannot start resource manager '%s' (Reason: %s).	The resource manager might not be available.
0x1100a929	ADMIN	system.moab	WARN	MWM_STDOUT_FAIL	Request succeeded with no stdout. stderr= '%s'.	The standard out might not have been specified.
0x1100a92a	ADMIN	system.moab	WARN	MWM_UNSUPPORTED_REQ_ATTR	Unsupported req attribute '%s'.	The attribute is not one that can be set.
0x1100	ADMI	system.mo	WA	MWM_UNSUPPORT	Unsupported general attribute '%s'.	The attribute is not one that

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a92b	N	ab	RN	ED_GENERAL_ATTR		can be set.
0x1100a92e	ADMIN	system.moab	WARN	MWM_CREATE_ACCOUNT	Unable to create account '%s' on the account manager.	Verify that the account manager is running.
0x1100a92f	ADMIN	system.moab	WARN	MWM_QUERY_ACCOUNT	Unable to query account '%s' on the account manager.	Verify that the account name is correct.
0x1100a930	ADMIN	system.moab	WARN	MWM_ACCOUNT_ADD_USER	Unable to add user '%s' to account '%s' on the account manager.	Verify that the account name is correct.
0x1100a931	ADMIN	system.moab	WARN	MWM_ACCOUNT_DEPOSIT	Unable to deposit '%s' credits to account '%s' on the account manager.	Verify that the account name is correct.
0x1100a932	ADMIN	system.moab	WARN	MWM_ALLOCATE_REQ	Unable to allocate requirement '%s' using NAllocPolicy '%s' (%s).	The system might be low on memory.
0x1100a933	ADMIN	system.moab	WARN	MWM_VALID_STAT_DATA	Unable to generate valid statistic data for external query.	The system might be low on memory.
0x1100a934	ADMIN	system.moab	WARN	MWM_JOB_QOS_REQUEST	Job '%s' cannot request QOS '%s').	The requested QOS is not available for the job.
0x1	AD	syste	W	MWM_GRES_	GRES overflow.	Unable to add

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a938	MIN	m.moab	ARN	OVERFLOW		another GRES.
0x1100a939	ADMIN	system.moab	WARN	MWM_RM_NO_RESOURCES	The resource manager '%s' is reporting no resources.	Check the nodes on the resource manager.
0x1100a93d	ADMIN	system.moab	WARN	MWM_EMPTY_FILE	File '%s' is empty.	Check the file specified.
0x1100a93f	ADMIN	system.moab	WARN	MWM_PREEMPT_NONACTIVE_JOB	Cannot preempt non-active job '%s' (state: '%s' estate: '%s').	The job must currently be active to preempt it.
0x1100a940	ADMIN	system.moab	WARN	MWM_REQUEUE_NONSTARTABLE_JOB	Cannot requeue non-startable job '%s' (canceling instead).	The job could not be requeued.
0x1100a941	ADMIN	system.moab	WARN	MWM_NODE_RACK_VALUE	Invalid rack value '%s' specified for node %s (must be digit).	Check the value of the rack parameter.
0x1100a942	ADMIN	system.moab	WARN	MWM_ENCODE_JOB_MESSAGE	Cannot encode job message.	Check the value of the rack parameter.
0x1100a943	ADMIN	system.moab	WARN	MWM_INVALID_PROFILECOUNT_VAL	Invalid PROFILECOUNT specified, modified internally to %s (see documentation).	The PROFILECOUNT value is invalid. Moab uses the default value

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						instead.
0x1100a944	ADMIN	system.moab	WARN	MWM_FAIRSHARE_FILE	Cannot load fairshare file '%s' for slot %s.	Check for the existence of the fairshare file in the file system.
0x1100a945	ADMIN	system.moab	WARN	MWM_REQ_ATTR_ALREADY_SET	Requirement attribute %s '%s' is already set.	Check the attribute setting in the configuration file.
0x1100a946	ADMIN	system.moab	WARN	MWM_ZERO_START_TIME	StartTime set to zero for reservation on job '%s'.	Check the start time for the specified job.
0x1100a947	ADMIN	system.moab	WARN	MWM_EXISTING_RESERVATION	Reservation created for reserved job '%s' (existing reservation '%s' deleted).	Only one reservation can exist at a time for the reserved job.
0x1100a948	ADMIN	system.moab	WARN	MWM_CANNOT_PARSE_REQ_LINE	Cannot parse requirement line for job '%s'.	The syntax of the requirement line is incorrect.
0x1100a949	ADMIN	system.moab	WARN	MWM_UNKNOWN_RESOURCE_TYPE	Unknown resource type '%s' for job '%s'.	Check the documentation for valid resource types.
0x1100a94a	ADMIN	system.moab	WARN	MWM_UNKNOWN_TRANSACTION_ATTR	Unknown transaction attribute '%s'.	Check the documentation for valid transaction attributes.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100a94c	ADMIN	system.moab	WARN	MWM_ID_MANAGER_DOWN	The identity manager is down.	Check the status of the identity manager.
0x1100a94d	ADMIN	system.moab	WARN	MWM_PROCESS_ID_LINE	Unable to process the ID line '%s'.	Check the syntax of the attribute/value pairs.
0x1100a94e	ADMIN	system.moab	WARN	MWM_JOB_ID_MISSING	Unable to locate the job ID for a job submitted to the resource manager.	Check the job being submitted`.
0x1100a950	ADMIN	system.moab	WARN	MWM_EMPTY_NODELIST	The nodelist is empty for reservation '%s'.	Reservations should include a node list.
0x1100a951	ADMIN	system.moab	WARN	MWM_FAIRSHARE_PAL	Fairshare does not allow specified PAL (%s).	The fairshare algorithm is reverting to the original PAL.
0x1100a952	ADMIN	system.moab	WARN	MWM_JOB_START_TIME	Cannot find earliest start time for job '%s'.	Resources needed to run the job may never be available.
0x1100a953	ADMIN	system.moab	WARN	MWM_NODE_MODIFY	Cannot modify node '%s' Error(%s).	The node could not be modified.
0x1100a95	ADMIN	system.moab	WAR	MWM_TRIGGER_RSV_CREATE	Unable to create a trigger reservation.	Check the reservation time, nodes,

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
4			N			and account.
0x1100a955	ADMIN	system.moab	WARN	MWM_MISSING_TRIGGER	Trigger '%s' with PID '%s' does not exist--completing!	The process might have already completed.
0x1100a956	ADMIN	system.moab	WARN	MWM_EMPTY_HOSTLIST	The hostlist is empty for reservation.	Reservations should include a hostlist.
0x1100a957	ADMIN	system.moab	WARN	MWM_RSV_POLICY_VIOLATION	Unable to create requested reservation due to a policy violation (%s).	Reservations must conform to existing policies.
0x1100a958	ADMIN	system.moab	WARN	MWM_RSV_CREATE_FAILURE	Unable to create requested reservation at time %s (%s).	Resources are unavailable at requested time.
0x1100a959	ADMIN	system.moab	WARN	MWM_RSV_OWNER	Cannot process owner '%s' for standing reservation '%s' (%s).	Consult the error message.
0x1100a95a	ADMIN	system.moab	WARN	MWM_RSV_PARTIAL	Partial reservation %s reserved %s of %s procs in partition '%s' to start in %s at (%s) %s.	Entire reservation could not be filled.
0x1100a95c	ADMIN	system.moab	WARN	MWM_RSV_NEGATIVE_JOBCOUNT	Reservation %s jobcount is %s, should not decrement less than 0.	JobCount cannot be negative.
0x1100a95	ADMIN	system.moab	WAR	MWM_EMPTY_REQ_	Req node list empty for job %s:%s in state %s (job nodelist copied to req	Job should include a req node list.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
d			N	NODELIST	nodelist).	
0x1100a95e	ADMIN	system.moab	WARN	MWM_TASK_ALLOCATION_INFO	Cannot locate task allocation info for job %s:%s in state %s.	Job should include a task list.
0x1100a95f	ADMIN	system.moab	WARN	MWM_INVALID_THREADPOOL_SIZE	Invalid ThreadPoolSize '%s' (must be a non-negative integer no larger than %s).	Check the size for a valid value.
0x1100a960	ADMIN	system.moab	WARN	MWM_INVALID_QUEUE_TIME	Job '%s' has invalid system queue time (SQ: %s > ST: %s).	Check the job queue time value.
0x1100a961	ADMIN	system.moab	WARN	MWM_NO_WCLIMIT	Job '%s' has no WCLimit specified.	Check the job for the correct value.
0x1100a962	ADMIN	system.moab	WARN	MWM_AM_INVALID_PROTOCOL	Invalid protocol '%s' specified for account manager '%s'.	Communication with the account manager must be over a supported protocol.
0x1100a963	ADMIN	system.moab	WARN	MWM_NO_POWER_INTERFACE	No external power interface - cannot set power state '%s' on node '%s%s%s'.	Cannot set the power state on the node without a power interface.
0x1100a96	ADMIN	system.moab	WAR	MWM_JOB_STARTED_ON_	Job '%s' started externally: (rc: %s; errmsg: '%s'; Tasklist: '%s').	Two or more resource managers are

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
4			N	ANOTHER_RM		running side-by-side and the job is already running on one of them.
0x1100a965	ADMIN	system.moab	WARN	MWM_COMMAND_FAILED_CHILD_PROCESS	Job submit request failed with child process status code='%s', stderr='%s', stdout='%s', EMsg='%s'.	Review the status code and error message for more information.
0x1100a967	ADMIN	system.moab	WARN	MWM_AM_REGISTER_JOB	Unable to register job creation with account manager for job '%s', reason: '%s'.	Check the status of the account manager.
0x1100a968	ADMIN	system.moab	WARN	MWM_AM_DEPRECATED_PARAMETER	Use of the '%s' parameter is deprecated. %s	Check the documentation for the new parameter syntax.
0x1100a969	ADMIN	system.moab	WARN	MWM_AM_INVALID_ACTION	Invalid action '%s' specified in '%s' for account manager '%s'.	Check the documentation for valid actions for the account manager.
0x1100a9bf	ADMIN	system.moab	WARN	MWM_ERASING_JOB	Erasing job '%s' by address.	The specified job could not be found by name. The entire job table was searched to find the matching job.
0x1	AD	syste	W	MWM_JOB_	Job '%s' has invalid task	The task layout

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a9e1	MIN	m.moab	ARN	INVALID_TASK_LAYOUT	layout (TPN:%s * N:%s != T:%s).	does not compute.
0x1100a9e2	ADMIN	system.moab	WARN	MWM_JOB_ACCESS_QOS	Job '%s' does not have access to QOS '%s' (QAL: %s).	The QoS is not accessible from the job.
0x1100a9f8	ADMIN	system.moab	WARN	MWM_DUPLICATE_SYSTEMJID	Duplicate SystemJID '%s' [JState: %s] found from RM '%s'.	The SystemJID must be unique.
0x1100a9fc	ADMIN	system.moab	WARN	MWM_CANNOT_PING_RM	Cannot ping RM '%s' because a file was not specified.	A file path to a valid file is needed.
0x1100aa07	ADMIN	system.moab	WARN	MWM_VM_CONTAINER_NODE	Cannot find or add container node '%s' for VM '%s'.	The node could not be found.
0x1100aa4b	ADMIN	system.moab	WARN	MWM_DATASTAGING_DYNAMIC_WALLTIME_CALCULATION_FAILURE	Failed to calculate dynamic walltime for data staging system job '%s'.	Unlimited walltime will be used.
0x1100aa53	ADMIN	system.moab	WARN	MWM_PBS_JOB_FULL_REPORT_TIME_MISMATCH	PBS server (%s) is configured with a job_full_report_time (%s) less than Moab's RMPollInterval (%s). This could lead to incomplete job information.	Moab/TORQUE configuration mismatch.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x1100aa54	ADMIN	system.moab	WARN	MWM_BACKLOGCOMPLETIONTIME_NEEDS_ENABLEPROFILING	BacklogCompletionTime cannot be calculated; EnableProfiling must be enabled on the QOS to gather stats.	BacklogCompletionTime needs EnableProfiling enabled.
0x1100aa62	ADMIN	system.moab	WARN	MWM_NATIVE_RM_OVERRIDE	RM '%s', of type '%s' is overriding default '%s' operation with configured 'native' call.	A non-native RM that is configured with native calls can override the default functions.
0x1100e710	INTERNAL	system.moab	WARN	MWM_INVALID_ARG_VALUE	Invalid arguments passed to this function.	One or more arguments passed to this function were not valid. This is an internal error logged for informational purposes.
0x1100e72b	INTERNAL	system.moab	WARN	MWM_SEND_SENT_NO_DATA	No data was sent to the socket when it should have been.	The send() system call reported no data was sent when data should have been sent.
0x1100e72e	INTERNAL	system.moab	WARN	MWM_SOCKET_BLOCKED_UNEXPECTEDLY	Read operations on the socket were blocked when it should have been available.	A socket operation reported that the operation was blocked. Previous

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						information indicated that this operation should have been available.
0x1100e73f	INTERNAL	system.moab	WARN	MWM_MUTEX_LOCK	Cannot lock mutex semaphore using pthread_mutex_lock().	This is an operating system call problem.
0x1100e740	INTERNAL	system.moab	WARN	MWM_MUTEX_UNLOCK	Cannot unlock mutex semaphore using pthread_mutex_unlock().	This is an operating system call problem.
0x1100e771	INTERNAL	system.moab	WARN	MWM_THREAD_DB_INIT	Thread %s attempting to re-initialize database info struct.	Internal error condition.
0x1100e84b	INTERNAL	system.moab	WARN	MWM_CORRUPT_COMMAND_RECEIVED	Corrupt command '%s' received.	The communication packet received from a Moab client command is malformed.
0x1100e919	INTERNAL	system.moab	WARN	MWM_CHECKPOINT_NO_XML	The checkpoint data does not contain XML.	Internal error.
0x1100e91a	INTERNAL	system.moab	WARN	MWM_CHECKPOINT_INVALID_XML	The checkpoint data does not contain valid XML (%s).	Internal error.
0x1	INT	syste	W	MWM_	Unable to update a	Internal error.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100e91b	ERNAL	m.moab	ARN	CHECKPOINT_UPDATE_RSV_FROM_XML	reservation from checkpoint XML.	
0x1100e927	INTERNAL	system.moab	WARN	MWM_JOB_ATTR_TO_STRING	Job attribute '%s' not yet translated to string value.	Internal warning.
0x1100e92c	INTERNAL	system.moab	WARN	MWM_MISSING_STATUS_CODE	The status code was missing from the S3 response.	This is an internal error.
0x1100e92d	INTERNAL	system.moab	WARN	MWM_MISSING_STATUS_VALUE	The status value was missing from the S3 response.	This is an internal error.
0x1100e93a	INTERNAL	system.moab	WARN	MWM_SIMULATION_NO_JOBS	No jobs loaded in simulation.	Internal simulation error.
0x1100e93b	INTERNAL	system.moab	WARN	MWM_SIMULATION_JOB_DETECTED_TRACEBUFFER	Job '%s' previously detected in tracefile (MJobTraceBuffer[%s]/JC: %s; IT: %s).	Internal simulation error.
0x1100e93c	INTERNAL	system.moab	WARN	MWM_SIMULATION_JOB_DETECTED	Job '%s' previously detected in tracefile (Job/JC: %s; IT: %s).	Internal simulation error.
0x1100e93	INTERNA	system.moab	WAR	MWM_READ_COMMAND_	Cannot read output of command '%s'.	This is an internal communication

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
e	L		N	OUTPUT		s error.
0x1100e94b	INTERNAL	system.moab	WARN	MWM_THREAD_TIMEOUT	Thread %s killed (%s micro-second time out reached).	This is an internal issue.
0x1100e94f	INTERNAL	system.moab	WARN	MWM_INVALID_XML_RM	Invalid XML data for resource manager '%s'.	Check the XML syntax.
0x21000067	USER	domain.lifecycle	ERROR	MWM_JOB_END_FAILED	Job %s failed at %s. %s	The job finished unsuccessfully.
0x210000ca	USER	domain.lifecycle	ERROR	MWM_NODE_EVAC_VMS_ERROR	Error evacuating VMs off node %s. %s	There was an error while attempting to evacuate the VMs off the node.
0x21002882	USER	system.moab	ERROR	MWM_NODE_MODIFY_FAILURE	Cannot modify node state of '%s' Error(%s).	The node state could not be modified.
0x21002a1a	USER	system.moab	ERROR	MWM_DEPRECATED_RM_FEATURE	RM flag SUBMITJOBSASROOT not supported with this version, %s. Must be >= 2.4.8.	The resource manager version should be updated to get support for this feature.
0x2100	ADMI	system.mo	ER	MWM_TESTING_	Testing with argument1: %s. and argument2: %s.	Internal error for testing

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
8003	N	ab	ROR	ERROR		diagnostics.
0x21008263	ADMIN	domain.lifecycle	ERROR	MWM_VC_SCHEDULE_FAILURE	Failed to schedule virtual container '%s'.	This is an internal error.
0x21008388	ADMIN	system.moab	ERROR	MWM_INVALID_LICENSE_USE_ERROR	Use of %s requires license with %s enabled	Requested feature must be licensed. Please contact your sales representative at Adaptive Computing for assistance.
0x2100a718	ADMIN	system.moab	ERROR	MWM_CANNOT_FORK_ERROR	Cannot fork the process, errno: %s (%s).	The fork() system call failed. Use the errno and associated message to determine possible causes.
0x2100a719	ADMIN	system.moab	ERROR	MWM_CANNOT_EXEC_PROGRAM	Cannot exec action '%s', errno: %s (%s).	The exec() system call failed to execute the command. This might be because the command does not exist or the permissions do not allow it to

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						be run. Use the errno and associated message to determine possible causes.
0x2100a71b	ADMIN	system.moab	ERROR	MWM_CANNOT_CHOWN_FILE	Failure changing ownership of file: '%s' to uid:'%s', gid:'%s', errno: %s (%s).	The chown() system call failed. Use the errno and associated message to determine possible causes.
0x2100a71f	ADMIN	system.moab	ERROR	MWM_PIPE_READ_FAILED	Failed to read pipe on command '%s', errno: %s (%s).	The fread() system call failed. Use the errno and associated message to determine possible causes.
0x2100a720	ADMIN	system.moab	ERROR	MWM_CANNOT_READ_FILE	Cannot read file '%s', errno: %s (%s).	The fread() system call failed. Use the errno and associated message to determine possible causes.
0x2100a72	ADMIN	system.moab	ERR	MWM_CANNOT_WRITE_TO_	Failure writing to file, errno: %s (%s).	The write() system call failed. Use the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
1			OR	FILE		errno and associated message to determine possible causes.
0x2100a725	ADMIN	system.moab	ERROR	MWM_CANNOT_GET_HOSTNAME	Cannot get hostname '%s', errno: %s (%s).	The gethostname() system call failed. Use the errno and associated message to determine possible causes.
0x2100a727	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_SOCKET	Failure creating a socket, errno: %s (%s).	The socket() system call failed. Use the errno and associated message to determine possible causes.
0x2100a728	ADMIN	system.moab	ERROR	MWM_CANNOT_CONNECT_TO_HOST	Failure connecting to server '%s' on port %s, errno: %s (%s).	The connect() system call failed. Use the errno and associated message to determine possible causes.
0x2100a73	ADMIN	system.moab	ERR	MWM_EPOCH_FAIL	Epoch Fail, time: '%s' cannot be converted to an epoch time, errno: %s (%s).	The mktime() system call failed. Use the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0			OR			errno and associated message to determine possible causes.
0x2100a731	ADMIN	system.moab	ERROR	MWM_MEMORY_ALLOCATION_FAILURE_MALLOC	Failure allocating memory (malloc), allocating '%s' bytes, errno: %s (%s).	The malloc() system call failed. Use the errno and associated message to determine possible causes.
0x2100a732	ADMIN	system.moab	ERROR	MWM_MEMORY_ALLOCATION_FAILURE_CALLOC	Failure allocating memory (calloc), allocating '%s' elements of size '%s' bytes, errno: %s (%s) in file %s:%s.	The calloc() system call failed. Use the errno and associated message to determine possible causes.
0x2100a733	ADMIN	system.moab	ERROR	MWM_MEMORY_ALLOCATION_FAILURE_REALLOC	Failure allocating memory (realloc), allocating '%s' bytes, errno: %s (%s).	The realloc() system call failed. Use the errno and associated message to determine possible causes.
0x2100a734	ADMIN	system.moab	ERRO	MWM_CANNOT_DUPLICATE_STRING	Failure duplicating string, errno: %s (%s).	The strdup() system call failed. Use the errno and

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
			R			associated message to determine possible causes.
0x2100a735	ADMIN	system.moab	ERROR	MWM_CANNOT_CHANGE_PROCESS_GROUP	Failure changing process group, errno: %s (%s).	The setpgrp() system call failed. Use the errno and associated message to determine possible causes.
0x2100a736	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_THREAD	Failure creating thread: '%s', errno: %s (%s).	The pthread_create() system call failed. Use the errno and associated message to determine possible causes.
0x2100a737	ADMIN	system.moab	ERROR	MWM_CANNOT_TRUNCATE_FILE	Failure truncating a file '%s', errno: %s (%s).	The truncate() system call failed. Use the errno and associated message to determine possible causes.
0x2100a739	ADMIN	system.moab	ERROR	MWM_PIPE_OPEN_FAILED	Failed to open pipe on command '%s', errno: %s (%s)	The popen() system call failed. Use the errno and associated

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						message to determine possible causes.
0x2100a73d	ADMIN	system.moab	ERROR	MWM_CHANGE_DIR_FAILURE	OS call to change directory to '%s' failed errno: %s (%s).	The chdir() system call failed. Use the errno and associated message to determine possible causes.
0x2100a74a	ADMIN	system.moab	ERROR	MWM_CANNOT_LOCK_MOAB_PID_FILE	Cannot lock the PID file '%s'. Is Moab already running?	Moab tries to ensure that only one instance of itself is running. In the default configuration it will exit if it cannot obtain a lock.
0x2100a74e	ADMIN	system.moab	ERROR	MWM_CONFIG_FILE_NOT_FOUND_ERROR	Cannot locate configuration file in any predetermined location.	Moab cannot find the configuration file. Verify that it is present and installed in a proper location.
0x2100a757	ADMIN	system.moab	ERROR	MWM_MWS_RM_CONFIGURATION	The resource manager with Moab Web Services (%s) does not have a base URL, username, and password configured.	Correctly configure the Moab Web Services resource

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						manager.
0x2100a761	ADMIN	system.moab	ERROR	MWM_STRICT_INVALID_CONFIG_LINE	Error processing line #%s: %s - (%s).	Check the line number in the configuration file.
0x2100a768	ADMIN	system.moab	ERROR	MWM_MONGOSERVER_INITIALIZATION_FAILED	Failed to initialize connection to Mongo server '%s' RelicatSetName: '%s' SSLMode: '%s' SSL CA File: '%s'.	Failed to initialize connection to the configured MONGOSERVER. Check the following: (1) network connection to Mongo server; and (2) check MONGOUSER and MONGOPASSWORD parameters in moab-private.cfg.
0x2100a76a	ADMIN	system.moab	ERROR	MWM_MONGOSERVER_AUTHENTICATION_FAILURE	Failed to authenticate to Mongo server (%s).	Check user credentials.
0x2100a76b	ADMIN	system.moab	ERROR	MWM_MONGOSERVER_WRITE_FAILURE	Unable to write out transition object '%s'.	The BSON information is invalid or missing.
0x2	AD	syste	E	MWM_	The Mongo server is down.	Check the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a76c	MIN	m.moab	RROR	MONGOSERVER_DOWN		status of the server.
0x2100a76f	ADMIN	system.moab	ERROR	MWM_DB_CHECKPOINT_OBJECT	Unable to checkpoint object to the database (%s).	Make sure the database is running.
0x2100a773	ADMIN	system.moab	ERROR	MWM_BACKUP_SERVER_CONNECTION_FAILED	The system was unable to connect to the backup server %s (%s:%s).	Make sure the backup server's address is correct.
0x2100a776	ADMIN	system.moab	ERROR	MWM_CONNECTION_REFUSED	Connection to the server was refused (%s).	Primary server refused and no fallback server available.
0x2100a777	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_CONNECT	Cannot send request to %s:%s (%s may not be running).	Unable to connect to the scheduler program.
0x2100a779	ADMIN	system.moab	ERROR	MWM_CLIENT_MAX_CONNECTIONS_REACHED	Cannot accept connection number %s (transaction number %s) from '%s' (limit reached).	May need to increase the CLIENTMAXCONNECTIONS configuration setting.
0x2100a77a	ADMIN	system.moab	ERROR	MWM_SERVER_CONNECTION_FAILED	The system was unable to connect to the server %s:%s - %s.	Make sure the server's address is correct and it is running.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a77c	ADMIN	system.moab	ERROR	MWM_COMMUNICATION_ERROR	Communication error %s:%s (%s).	General error trying to communicate with the host.
0x2100a77d	ADMIN	system.moab	ERROR	MWM_CANNOT_PARSE_SERVER_RESPONSE_STATUS	Cannot parse server response (status).	The response sent from the server is malformed.
0x2100a77e	ADMIN	system.moab	ERROR	MWM_CANNOT_PARSE_SERVER_RESPONSE_DATA	Cannot parse server response (data).	The response sent from the server is malformed.
0x2100a77f	ADMIN	system.moab	ERROR	MWM_INVALID_FS_TARGET	Invalid type specified for FSTarget.	Fairshare target type is invalid.
0x2100a780	ADMIN	system.moab	ERROR	MWM_COULD_NOT_ADD_FS_TREE_NODE	Could not add fstree node %s.	Unable to add a node to the fairshare configuration tree.
0x2100a781	ADMIN	system.moab	ERROR	MWM_CANNOT_ADD_MANAGER_TO_FS_TREE	Could not add manager %s to fstree.	Unable to add a manager to the fairshare tree.
0x2100	ADMIN	system.moab	ER	MWM_CRED_	CredManager overflow, manager %s not added.	Credential Manager could

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a782			ROR	MANAGER_OVERFLOW		not add another manager.
0x2100a783	ADMIN	system.moab	ERROR	MWM_CRED_MANAGER_OVERFLOW_CHILD	CredManager overflow while adding managers to child in fstree.	Fairshare tree configuration problem.
0x2100a786	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_SELECT_TASKS_FOR_JOB	Cannot select tasks for job %s. (%s)	Cannot select a node list that matches the requirements for this job. This might not be serious since multiple passes may occur.
0x2100a787	ADMIN	system.moab	ERROR	MWM_CANNOT_GET_TASK_ON_RESERVATION	Cannot get tasks on (ERR: %s/no reservation/iteration %s).	Cannot select tasks that meet the requirements.
0x2100a78a	ADMIN	system.moab	ERROR	MWM_BEST_VAL_ACHIEVED_BUT_SCHEDULE_EMPTY	BestVal %s achieved but schedule is empty.	Best value has been set, but the schedule is empty.
0x2100a78b	ADMIN	system.moab	ERROR	MWM_JOB_SCHEDULING_FAILURE_NO_RESERVATION	Scheduling failure %s (policy violation/no reservation) iteration: %s. (%s)	The job was not scheduled because no reservations are available.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a78c	ADMIN	system.moab	ERROR	MWM_UNSUPPORTED_SERVICE	Service '%s' (%s) not supported.	A request for an unsupported service was sent.
0x2100a78d	ADMIN	system.moab	ERROR	MWM_INVALID_CLASS_HOST_EXPRESSION	Invalid class host expression received (%s) : %s.	Failed to expand the class's host pattern to a list.
0x2100a78e	ADMIN	system.moab	ERROR	MWM_TOO_MANY_COALLOCATION_REQUESTS	Too many co-allocation requests (%s > %s).	Too many co-allocation requests were received.
0x2100a78f	ADMIN	system.moab	ERROR	MWM_INVALID_JOBID_COUNTER	Min Job ID '%s' must be less than Max Job ID '%s'.	Invalid job ID was encountered.
0x2100a790	ADMIN	system.moab	ERROR	MWM_PARAMETER_NOT_HANDLED	Parameter[%s] '%s' not handled.	The specified parameter was not handled due to an unknown format.
0x2100a791	ADMIN	system.moab	ERROR	MWM_CIRCULAR_JOB_DEPENDENCY	Job cannot be dependent on itself.	The job is trying to use itself as a dependency, which creates a circular dependency and is invalid.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a792	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_AM	Cannot create AM %s.	Could not create account manager object.
0x2100a793	ADMIN	system.moab	ERROR	MWM_INVALID_FLUSH_INTERVAL	%s for AM %s.	An invalid flush interval has been entered.
0x2100a794	ADMIN	system.moab	ERROR	MWM_FAILED_SERVER_AUTH	Unable to authenticate server.	The server could not be authenticated.
0x2100a795	ADMIN	system.moab	ERROR	MWM_NO_QUOTE	No quote output provided in response.	No quote output provided in response.
0x2100a796	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_PARSE_XML	Unable to parse XML (%s): %s.	Unable to parse XML data.
0x2100a797	ADMIN	system.moab	ERROR	MWM_INVALID_QUOTE	Invalid quote amount (%s).	Quote is invalid.
0x2100a798	ADMIN	system.moab	ERROR	MWM_RECURRING_COST	Unable to determine recurring cost.	Unable to determine recurring cost.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a799	ADMIN	system.moab	ERROR	MWM_AVAILABLE_PORT_NOT_FOUND	Cannot locate an available port for listening.	After trying to bind to a large number of ports, none were found to be available. Check network socket status for saturation.
0x2100a79a	ADMIN	system.moab	ERROR	MWM_CANNOT_RESOLVE_IP_FROM_HOSTNAME	Cannot resolve IP address from hostname '%s', getaddrinfo() rc: %s (%s).	There is a failure matching an IP address to a hostname. Check DNS, /etc/hosts or applicable nameservice.
0x2100a79b	ADMIN	system.moab	ERROR	MWM_UNKNOWN_CHECKPOINT_TYPE	Unexpected checkpoint type, %s.	Unknown checkpoint type while reading from the file.
0x2100a79c	ADMIN	system.moab	ERROR	MWM_CHECKPOINT_FILE_LINE_NOT_HANDLED	Line '%s' not handled in checkPoint file '%s'.	Please contact Adaptive Computing for assistance.
0x2100a79d	ADMIN	system.moab	ERROR	MWM_CANNOT_ADD_DEFAULT_GROUP	Cannot add default group.	Default group cannot be added.
0x2100a79	ADMIN	system.moab	ERR	MWM_CANNOT_ADD_GROUP	Cannot add group %s.	Group cannot be added.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
e			OR			
0x2100a79f	ADMIN	system.moab	ERROR	MWM_ACCOUNT_NOT_ACCESSIBLE_BY_JOB	Account '%s' is not accessible by job '%s'.	The job is not authorized to run under the listed account.
0x2100a7a0	ADMIN	system.moab	ERROR	MWM_CANNOT_DETERMINE_DEFAULT_ACCOUNT	Unable to determine default account for job '%s', user '%s'.	There is not a default account type for this job.
0x2100a7a1	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_RESERVATION	Cannot create reservation for job '%s'.	Failed to create reservation for job.
0x2100a7a2	ADMIN	system.moab	ERROR	MWM_INVALID_NODELIST_BAD_TASKCOUNT	Invalid nodelist for job %s:%s (inadequate taskcount, %s < %s).	Invalid node list due to inadequate task count.
0x2100a7a3	ADMIN	system.moab	ERROR	MWM_INVALID_NODELIST_BAD_NODECOUNT	Invalid nodelist for job %s:%s (inadequate nodecount, %s < %s).	Invalid node list due to inadequate node count.
0x2100a7a4	ADMIN	system.moab	ERROR	MWM_INVALID_ALLOCATION_POLICY	Invalid allocation policy (%s).	Invalid allocation policy.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a7a5	ADMIN	system.moab	ERROR	MWM_NO_MEMORY_FOR_ALLOCPARTITION_VARIABLE	Cannot set ALLOCPARTITION variable for job %s (no memory).	No memory remaining to create job variable.
0x2100a7a6	ADMIN	system.moab	ERROR	MWM_BASIL_RSVID_NOT_FOUND	Cannot locate BASIL RSVID (job 'ALLOCPARTITION' variable) that was just created.	Cannot locate BASIL reservation ID stored in the ALLOCPARTITION variable.
0x2100a7a7	ADMIN	system.moab	ERROR	MWM_JOB_ADD_CLASS_ATTR	Cannot add class for job %s (Class: %s).	Unable to add a class requirement attribute to a job.
0x2100a7a8	ADMIN	system.moab	ERROR	MWM_JOB_ADD_DRM_ATTR	Cannot set destination RM for job %s (RM: %s).	Unable to add a destination resource manager attribute to a job.
0x2100a7a9	ADMIN	system.moab	ERROR	MWM_JOB_FLAGS_INVALID_SOURCE	Attempting to set job flags from invalid format.	Job flags must be created using documented formats.
0x2100a7aa	ADMIN	system.moab	ERROR	MWM_SET_SIZE_ON_NONEXISTENT_REQ	Requirement must be created before size is set.	Unable to set the size of an unallocated requirement.
0x2	AD	syste	E	MWM_ADD_	Cannot add group for job %s (Group: %s).	Unable to set a

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a7ab	MIN	m.moab	RROR	GROUP_TO_JOB_FAILURE		group attribute on a job.
0x2100a7ac	ADMIN	system.moab	ERROR	MWM_NULL_JOB_NAME	Cannot add an empty name as an alternate name attribute for job %s.	No value specified. Make sure the alternate job name has a value.
0x2100a7ad	ADMIN	system.moab	ERROR	MWM_SPACES_IN_JOB_NAME	Attempted to set a job name (%s) with space(s) for job %s.	A job name with space(s) was specified. Job names cannot contain embedded spaces.
0x2100a7ae	ADMIN	system.moab	ERROR	MWM_ADD_QOS_TO_JOB_FAILURE	Cannot add QOS for job %s (QOS: %s).	Unable to set a QOS attribute on a job.
0x2100a7af	ADMIN	system.moab	ERROR	MWM_ADD_SRM_TO_JOB_FAILURE	Cannot add Submit RM for job %s (RM: %s).	Unable to find the entered name as an available resource manager.
0x2100a7b0	ADMIN	system.moab	ERROR	MWM_ADD_VARIABLE_TO_JOB_FAILURE	Cannot set variable for job %s (no variable name specified).	Only variables with names can be added as a job attribute.
0x2	AD	syste	E	MWM_ADD_	Cannot add user for job %s (User: %s).	Unable to set a

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a7b1	MIN	m.moab	RROR	USER_TO_JOB_FAILURE		user attribute on a job.
0x2100a7b2	ADMIN	system.moab	ERROR	MWM_ADD_NODE_TO_JOB_FAILURE	Cannot add node for job %s (Node: %s).	Unable to set a node attribute on a job.
0x2100a7b3	ADMIN	system.moab	ERROR	MWM_ADD_ACCOUNT_TO_JOB_FAILURE	Cannot add account for job %s (Name: %s).	Failed to add account to the job.
0x2100a7b5	ADMIN	system.moab	ERROR	MWM_INVALID_TIME_STRING	Invalid format for time specification: '%s'.	A string that describes a time cannot be parsed because the format is wrong, or the values are out of range.
0x2100a7b6	ADMIN	system.moab	ERROR	MWM_CANNOT_FIND_ARRAY_JOB	Cannot find array job at index %s for job '%s'.	Array job is missing.
0x2100a7b7	ADMIN	system.moab	ERROR	MWM_JOB_BUFFER_FULL	Job buffer is full (ignoring job '%s').	Ignoring job since job buffer is full. Try increasing the value specified for the MAXJOB parameter.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a7b8	ADMIN	system.moab	ERROR	MWM_CANNOT_FIND_MASTER_JOB	Cannot find master job (%s) for job '%s'; job array slot limits may not be enforced.	Cannot find the master job that is associated with a job array.
0x2100a7b9	ADMIN	system.moab	ERROR	MWM_INVALID_ACTION_STRING	The action string (%s) is invalid.	The format of the action string is '<operation type>:<operation ID>:<operation action>' Example: job:145+146+147:cancel where 145,146 and 147 are job IDs.
0x2100a7ba	ADMIN	system.moab	ERROR	MWM_INVALID_OBJECT_TYPE	The object type %s is invalid.	The format of the action string is '<operation type>:<operation ID>:<operation action>' Example: job:145+146+147:cancel where 145,146 and 147 are job IDs.
0x2100a7bb	ADMIN	system.moab	ERROR	MWM_JOB_NOT_FOUND	Unable to locate job %s.	The named job was not located in the system.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a7bc	ADMIN	system.moab	ERROR	MWM_JOB_IN_BAD_STATE_FOR_COMPLETE	Completed trigger action is specified for job %s but it is in an invalid state.	The job is not a system job and is not allowed to be started by the resource manager.
0x2100a7bd	ADMIN	system.moab	ERROR	MWM_JOB_CANNOT_BE_HELD	Job %s cannot be put into hold state.	The resource manager cannot hold the job, usually because the job is not in a state that can be held.
0x2100a7be	ADMIN	system.moab	ERROR	MWM_CANNOT_SET_TRIGVAR	Cannot set trigger variable on job %s.	The trigger variables on a job cannot be set.
0x2100a7bf	ADMIN	system.moab	ERROR	MWM_CANNOT_SET_REQATTR	Cannot set request attribute variable on job %s.	The request attribute variables on a job cannot be set.
0x2100a7c0	ADMIN	system.moab	ERROR	MWM_CANNOT_ADJUST_GRES	Cannot adjust generic resources for job %s.	The generic resources of the job could not be modified.
0x2100a7c1	ADMIN	system.moab	ERROR	MWM_INVALID_GRES_VALUE	Invalid value '%s' for GRes '%s' %s.	The value being set on the generic resource is not valid.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a7c2	ADMIN	system.moab	ERROR	MWM_CANNOT_MODIFY_ATTRIBUTE	Attribute %s cannot be modified for job %s.	The job's attribute could not be modified.
0x2100a7c3	ADMIN	system.moab	ERROR	MWM_COULD_NOT_SEND_SIGNAL	Signal %s could not be sent to job %s.	The resource manager was unable to send the signal to the job.
0x2100a7c4	ADMIN	system.moab	ERROR	MWM_COULD_NOT_START_JOB	Could not start job %s in %s.	The resource manager was unable to start the job.
0x2100a7c5	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_REQUEUE_JOB	Cannot requeue job %s.	The job could not be requeued.
0x2100a7c6	ADMIN	system.moab	ERROR	MWM_UNHANDLED_ACTION	The action %s was not handled.	The action was undefined in this function.
0x2100a7c7	ADMIN	system.moab	ERROR	MWM_UNRECOGNIZED_ATTRIBUTE	The attribute %s is not recognized.	The attribute is not in the lookup table.
0x2100a7c8	ADMIN	system.moab	ERROR	MWM_UNRECOGNIZED_JOB_ACTION	The job action %s is not recognized.	The job action is not in the lookup table.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a7c9	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_CANCEL_JOB	Job %s could not be canceled.	The job could not be canceled.
0x2100a7ca	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_HOLD_JOB	Job %s could not be held in.	The job was unable to be put into a hold state.
0x2100a7cb	ADMIN	system.moab	ERROR	MWM_INVALID_PBS_SBINDIR	Invalid SBINDIR specified (%s).	Check paths for the directory containing pbs_iff.
0x2100a7cc	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_CONNECT_PBS_SRVR	Cannot connect to PBS server '%s'; rc: %s (pbs_errno=%s, '%s').	Make sure the pbs_server process is running.
0x2100a7cd	ADMIN	system.moab	ERROR	MWM_UNABLE_GET_SRVR_INFO	Cannot get server info: %s.	Make sure that the pbs_server process is running.
0x2100a7ce	ADMIN	system.moab	ERROR	MWM_UNABLE_LOAD_SRVR_INFO	Cannot load PBS server info: %s.	Make sure that the pbs_server process is running.
0x2100a7cf	ADMIN	system.moab	ERROR	MWM_UNABLE_LOAD_PBS_CLUSTER	Cannot load PBS cluster info: %s.	Make sure that the pbs_server process is running.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a7d0	ADMIN	system.moab	ERROR	MWM_UNABLE_LOAD_PBS_WORKLOAD	Cannot load PBS workload info: %s.	Make sure that the pbs_server process is running.
0x2100a7d1	ADMIN	system.moab	ERROR	MWM_UNABLE_LOAD_PBS_QUEUE	Cannot load PBS queue info: %s.	Make sure the path to the queue configuration is accessible by Moab.
0x2100a7d3	ADMIN	system.moab	ERROR	MWM_UNABLE_PROCESS_NODE_INFO	Cannot process node info.	Make sure the resource manager is running.
0x2100a7d4	ADMIN	system.moab	ERROR	MWM_NODE_BUFFER_FULL	Node buffer is full (ignoring node '%s').	Try increasing the node buffer.
0x2100a7d5	ADMIN	system.moab	ERROR	MWM_JOB_CANNOT_START	Job '%s' cannot be started: (cannot generate Tasklist).	Check the PBS server log to see reason of failure.
0x2100a7d6	ADMIN	system.moab	ERROR	MWM_JOB_CANNOT_START_TASK_EMPTY	Job '%s' cannot be started: (empty Tasklist).	Check the PBS server log to see reason of failure.
0x2100a7d7	ADMIN	system.moab	ERR	MWM_UNABLE_SET_NODE_COUNT	Cannot set nodecount for job '%s' - %s.	Check the PBS server log to see reason of failure.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
			OR			
0x2100a7d8	ADMIN	system.moab	ERROR	MWM_UNABLE_SET_WALLTIME	Cannot set walltime for job '%s' - %s.	Check the PBS server log to see reason of failure.
0x2100a7d9	ADMIN	system.moab	ERROR	MWM_UNABLE_SET_TASKLIST	Cannot set Tasklist for job '%s' - %s.	Check the PBS server log to see reason of failure.
0x2100a7da	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_START_JOB_RC	Job '%s' cannot be started: (rc: %s; errmsg: '%s'; Tasklist: '%s').	Check the PBS server log to see reason of failure.
0x2100a7db	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_SIGNAL_JOB	%s' cannot be signaled: %s.	Check the PBS server log to see reason of failure.
0x2100a7dc	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_SUSPEND_JOB	Job '%s' cannot be suspended: %s.	Check the PBS server log to see reason of failure.
0x2100a7dd	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_RESUME_JOB	Job '%s' cannot be resumed: %s.	Check the PBS server log to see reason of failure.
0x2	AD	syste	E	MWM_	Failed to find/add %s	Failure to

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a7de	MIN	m.moab	RROR	UNABLE_TO_FIND_RESOURCE	generic resource.	find/add GPUs/MICs to the global GRES/MIC slots.
0x2100a7df	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_SET_CREDENTIALS	Cannot authenticate job '%s' (U: %s; G: %s; A: '%s').	Could not set the credentials on the job.
0x2100a7e0	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_REQUEUE	PBS job '%s' cannot be requeued (rc: %s; '%s').	Check the PBS server log to see reason of failure.
0x2100a7e1	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_CHECKPOINT	PBS job '%s' cannot be checkpointed (rc: %s; '%s').	Check the PBS server log to see reason of failure.
0x2100a7e2	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_RELEASE	PBS job '%s' cannot be released from hold (rc: %s; '%s').	Check the PBS server log to see reason of failure.
0x2100a7e5	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_FIND_ACCOUNT	Cannot find account for job %s (Name: %s).	Make sure the account exists.
0x2100a7e6	ADMIN	system.moab	ERROR	MWM_INVALID_ARGUMENT	Command '%s' args not handled.	An unsupported argument was used.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a7e7	ADMIN	system.moab	ERROR	MWM_INVALID_LOGDIR	LogDir '%s' is invalid.	Make sure that the path to the logs directory exists.
0x2100a7e8	ADMIN	system.moab	ERROR	MWM_INVALID_SPOOLDIR	SpoolDir '%s' is invalid.	Make sure that the path to the spool directory exists.
0x2100a7e9	ADMIN	system.moab	ERROR	MWM_INVALID_STATDIR	StatDir '%s' is invalid.	Make sure that the path to the stat directory exists.
0x2100a7ea	ADMIN	system.moab	ERROR	MWM_INVALID_TOOLSDIR	ToolsDir '%s' is invalid.	Make sure that the path to the tools directory exists.
0x2100a7eb	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_DAT_FILE	Cannot create/modify dat file: '%s'.	Moab encountered an error creating the dat file.
0x2100a7ec	ADMIN	system.moab	ERROR	MWM_FEATURE_NOT_AVAILABLE_IN_BUILD	The '%s' feature is not available in the build of Moab.	Moab can be configured with various features. The listed feature is not available in the binary being run.
0x2	AD	syste	E	MWM_	The '%s' feature is not	Moab can be

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a7ed	MIN	m.moab	RROR	FEATURE_NOT_AVAILABLE_WITH_LICENSE	enabled with the current Moab license.	licensed with various features. The listed feature is not available with the current license. Contact Adaptive Computing for more information.
0x2100a7ee	ADMIN	system.moab	ERROR	MWM_RESOURCE_LIMIT_EXCEEDED	The maximum number of '%s' (%s) has been reached.	Moab has certain resources that are limited. This error occurs when you have reached or exceeded those limits. Contact Adaptive Computing for more information.
0x2100a7f1	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_VM_MIGRATION_JOB	Failed to create migration job for VM %s.	The migration job was not created. Check MIGRATETEMPLATE on workflow and its trigger.
0x2100a7f2	ADMIN	system.moab	ERROR	MWM_CANNOT_OPEN_EXTENSION_INTERFACE	Cannot open extension interface socket on port %s.	There was a failure opening the HTTP extension service. This

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						feature will not work until the problem is corrected.
0x2100a7f3	ADMIN	system.moab	ERROR	MWM_JOB_USER_AUTHENTICATION	The system was unable to connect the given user to job %s (User: %s, Group: %s).	Check the credentials of the given user and/or group.
0x2100a7f4	ADMIN	system.moab	ERROR	MWM_JOB_AUTHENTICATION	The system was unable to authenticate the user connected with job %s (User: %s, Group: %s, Account %s) - %s.	Check the credentials of the given user and/or group.
0x2100a7f5	ADMIN	system.moab	ERROR	MWM_SEND_DATA_FAILED	The system was unable to send data to the server %s (%s:%s).	Make sure the server's address is correct and that the server is running.
0x2100a7f6	ADMIN	system.moab	ERROR	MWM_RECEIVE_DATA_FAILED	The system was unable to receive data from the server %s (%s:%s).	Make sure the server's address is correct and that the server is running.
0x2100a7f7	ADMIN	system.moab	ERROR	MWM_JOB_OVERLAP	Job '%s' overlaps an existing job.	Check the job being created for overlap.
0x2100a7f8	ADMIN	system.moab	ERR	MWM_JOB_CREATION	The system was unable to create job '%s'	Verify that the job being created is

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
			OR			correctly specified.
0x2100a7f9	ADMIN	system.moab	ERROR	MWM_MISSING_STATUS_ELEMENT	The status element was missing from the S3 response.	This is an internal error.
0x2100a7fa	ADMIN	system.moab	ERROR	MWM_VC_WORKFLOW_JOB	Virtual container '%s' was marked as workflow, but could not find job that created it.	This is an internal error.
0x2100a7fb	ADMIN	system.moab	ERROR	MWM_VC_COMBINE_JOBS	Failed to combine jobs in virtual container '%s'.	This is an internal error.
0x2100a7fc	ADMIN	system.moab	ERROR	MWM_VC_SCHEDULE_TIME_FAILURE	Failed to schedule virtual container '%s' for requested time.	This is an internal error.
0x2100a7fd	ADMIN	system.moab	ERROR	MWM_VC_RESERVATION_FAILURE	Failed to find a reservation for virtual container '%s'.	This is an internal error.
0x2100a7fe	ADMIN	system.moab	ERROR	MWM_VC_RESERVATION_CREATE_FAILURE	Failed to create a reservation for jobs in virtual container '%s'.	This is an internal error.
0x2	AD	syste	E	MWM_VC_	Requested resources are not	This is an

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
100a7ff	MIN	m.moab	RROR	RESOURCE_FAILURE	available at any time for virtual container '%s'.	internal error.
0x2100a800	ADMIN	system.moab	ERROR	MWM_NONEXISTING_JOB_USER	Job template %s requests non-existent user %s.	Make sure the user exists.
0x2100a801	ADMIN	system.moab	ERROR	MWM_NONEXISTING_JOB_GROUP	Job template %s requests non-existent group %s.	Make sure the group exists.
0x2100a802	ADMIN	system.moab	ERROR	MWM_NONEXISTING_JOB_QOS	Job template %s requests non-existent QoS %s.	Make sure the QoS exists.
0x2100a803	ADMIN	system.moab	ERROR	MWM_UNABLE_CREATE_CLASS	Unable to create class %s for job template %s.	Make sure the class exists.
0x2100a804	ADMIN	system.moab	ERROR	MWM_NONEXISTING_JOB_ACCOUNT	Job template %s requests non-existent account %s.	Make sure the account exists.
0x2100a805	ADMIN	system.moab	ERROR	MWM_INVALID_WALLTIME_SPECIFIED	Invalid walltime specification '%s.	Make sure the format for walltime is correct.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a806	ADMIN	system.moab	ERROR	MWM_UNABLE_PARSE_WIKI_STR	Cannot parse wiki string for job '%s'.	Make sure the format for wiki string is correct.
0x2100a807	ADMIN	system.moab	ERROR	MWM_MISSING_STATS_XML_ELEMENT	%s is not a valid template job stat child element.	Make sure there is a stats element in the XML.
0x2100a808	ADMIN	system.moab	ERROR	MWM_NULL_NODE_POINTER	Node pointer is NULL and cannot be used to find SMP node.	Node pointer is NULL and cannot be used to find SMP node by node.
0x2100a809	ADMIN	system.moab	ERROR	MWM_PINDEX_OUT_OF_RANGE	PIndex is less than -1, which is out of range.	PIndex must be greater than or equal to -1 to find a node by partition.
0x2100a80a	ADMIN	system.moab	ERROR	MWM_FEATURE_OUT_OF_RANGE	Feature is less than -1, which is out of range.	Feature must be greater than or equal to -1 to find a node by feature.
0x2100a80b	ADMIN	system.moab	ERROR	MWM_INCORRECT_ARG	Incorrect argument in %s: %s, %s,%s.	Name must point to a valid string, Feature must be greater than or equal to -1, and N must point to a valid node.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a80c	ADMIN	system.moab	ERROR	MWM_NODE_ALLOCATION_ERROR	Failed to allocate a node named %s.	Call to MUMalloc failed, system is probably low on memory.
0x2100a80d	ADMIN	system.moab	ERROR	MWM_FAILED_TO_APPEND_MSMPNODE	Failed to append smpnode %s to MSMPNodes.	The call to append the node to the array list failed, probably due to a low memory condition.
0x2100a80e	ADMIN	system.moab	ERROR	MWM_NULL_SMPNODE_POINTER	Cannot initialize node because pointer is NULL.	Call to MSMPNodeInitialize must have a valid pointer to a valid node.
0x2100a80f	ADMIN	system.moab	ERROR	MWM_NULL_SMPNODE_POINTER_IN_RESET	Cannot reset node because pointer is NULL.	Call to MSMPNodeResetStats must have a valid pointer to a valid node.
0x2100a810	ADMIN	system.moab	ERROR	MWM_RESET_NODE_FAILED	Call to MSMPNodeResetStats failed.	Call to MSMPNodeResetStats failed. The most likely cause is passing a NULL pointer to SMPNode.
0x2100a81	ADMIN	system.moab	ERR	MWM_FREE_NODE_FAILED	Call to free MSMPNodes failed.	Call to free MSMPNodes failed, most

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
1			OR			likely due to corrupted memory.
0x2100a812	ADMIN	system.moab	ERROR	MWM_NULL_NODE_IN_UPDATE	Node pointer in %s cannot be NULL.	Node pointer cannot be NULL when trying to update node.
0x2100a813	ADMIN	system.moab	ERROR	MWM_CANNOT_FIND_NODE	Unable to find SMP node with node %s.	Unable to find SMP node by node.
0x2100a814	ADMIN	system.moab	ERROR	MWM_EMPTY_NODE_LIST	Updating node from list with empty node list.	Updating node from node list must not be called with an empty node list.
0x2100a815	ADMIN	system.moab	ERROR	MWM_BAD_ARG_IN_FEASIBLE_JOB	Incorrect argument to function %s: %s, %s.	A parameter in the function was incorrect.
0x2100a816	ADMIN	system.moab	ERROR	MWM_CANNOT_FIND_INDEX_IN_LIST_FOR_FEATURE	Could not find index into NodeSetList for node feature %s.	Could not find index into NodeSetList for node feature.
0x2100a817	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_SR	Could not create standing reservation: %s.	Failed to create the named standing reservation.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a818	ADMIN	system.moab	ERROR	MWM_UNEXPECTED_STATISTICS_TYPE	Unexpected statistics type: %s.	Number is not a member of MMStatTypeEnum enumeration.
0x2100a819	ADMIN	system.moab	ERROR	MWM_CANNOT_PROCESS_VM_ATTRIBUTE	Cannot process VM attribute %s for VM %s.	Either AttrName or NodeName is not found in string.
0x2100a81a	ADMIN	system.moab	ERROR	MWM_CANNOT_FIND_NODE_FOR_VM	Cannot find node %s for VM %s.	The node does not exist or cannot be found.
0x2100a81b	ADMIN	system.moab	ERROR	MWM_CANNOT_LOAD_JOB	Cannot load job %s (state: %s).	There was an error creating a job in Moab that was reported by the resource manager.
0x2100a81c	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_CHECKPOINT_FILE_ENTRY	Cannot create checkpoint file entry.	There was an error writing a checkpoint file entry for the associated objects.
0x2100a81d	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_OBJECT_FROM_CHECKPOINT_FILE	Cannot create object from checkpoint file entry.	There was an error reading a checkpoint file entry for the associated objects.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a81e	ADMIN	system.moab	ERROR	MWM_TASKLIST_TOO_LARGE	The tasklist for job '%s' is too large (size = %s, growth = %s).	The system has a fixed maximum size for the task map for each job.
0x2100a81f	ADMIN	system.moab	ERROR	MWM_TASKLIST_MISSING	The tasklist for job '%s' is missing.	The system requires that each job has at least one task assigned.
0x2100a820	ADMIN	system.moab	ERROR	MWM_TASK_DISTRIBUTION_UNKNOWN	The system encountered an unknown type of task distribution (%s).	This is an internal error.
0x2100a822	ADMIN	system.moab	ERROR	MWM_INCOMPATIBLE_CHARGE_POLICY	Periodic charging disabled due to incompatible job charge policy (%s).	The job charge policy is undefined.
0x2100a823	ADMIN	system.moab	ERROR	MWM_INCOMPLETE_JOB_TEMPLATE_ACTION	The job template '%s' has an incomplete action specification.	Job templates must fully specify the action to be performed.
0x2100a824	ADMIN	system.moab	ERROR	MWM_INCOMPLETE_JOB_TEMPLATE_GENERIC	The job template '%s' has an incomplete generic system job specification.	Job templates must fully specify the generic system job.
0x2100a825	ADMIN	system.moab	ERR	MWM_DUPLICATE_JOB_	The job template '%s' has a job '%s' that requests an existing VMID.	Virtual machine IDs cannot be

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
			OR	TEMPLATE_VMID		shared across job templates.
0x2100a826	ADMIN	system.moab	ERROR	MWM_UNKNOWN_JOB_TEMPLATE_VMID	The requested VMID '%s' could not be found or already has a tracking job.	Virtual machine IDs can only be assigned to a single job.
0x2100a827	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_MODIFY_JOB	The job '%s' on account '%s' cannot be modified in the resource manager.	The job previously submitted to the resource manager cannot be modified.
0x2100a828	ADMIN	system.moab	ERROR	MWM_WORKFLOW_VC_FAILURE	The system failed to generate a workflow virtual container for job '%s'.	This is an internal error.
0x2100a829	ADMIN	system.moab	ERROR	MWM_CREATE_JOB_TEMPLATE_FAILURE	The system failed to create job template '%s'.	The job could not be created or one of its attributes could not be set.
0x2100a82a	ADMIN	system.moab	ERROR	MWM_VC_NOT_FOUND	The system could not find the virtual container for job '%s'.	This is an internal error.
0x2100a82b	ADMIN	system.moab	ERROR	MWM_JOB_MIGRATION_FAILED	The system failed to migrate a remote job (%s).	Make sure the resource manager has not been disabled.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a82c	ADMIN	system.moab	ERROR	MWM_JOB_START_XML_FAILURE	The system could not generate the command line needed to start job: '%s'.	The proper command line could not be derived from the XML structure.
0x2100a82d	ADMIN	system.moab	ERROR	MWM_JOB_START_FAILURE_RESPONSE	The system could not start job - Reason: '%s'.	The system was unable to start the job for the specified reason.
0x2100a82e	ADMIN	system.moab	ERROR	MWM_JOB_CANCEL_FAILURE_RESPONSE	The system could not cancel job - Reason: '%s'.	The system was unable to cancel the job for the specified reason.
0x2100a82f	ADMIN	system.moab	ERROR	MWM_JOB_SIGNAL_FAILURE_RESPONSE	The system could not signal job - Reason: '%s'.	The system was unable to signal the job for the specified reason.
0x2100a830	ADMIN	system.moab	ERROR	MWM_JOB_MODIFY_FAILURE_RESPONSE	The system could not modify job - Reason: '%s'.	The system was unable to modify the job for the specified reason.
0x2100a831	ADMIN	system.moab	ERROR	MWM_JOB_REQUEUE_FAILURE_RESPONSE	The system could not requeue job - Reason: '%s'.	The system was unable to requeue the job for the specified reason.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a832	ADMIN	system.moab	ERROR	MWM_SEND_EVENT_FAILURE	The system could send event '%s' to resource manager '%s' (%s).	The system was unable to send the event.
0x2100a833	ADMIN	system.moab	ERROR	MWM_UNEXPECTED_BACKFILL_POLICY	The system encountered an unexpected backfill policy '%s' (using '%s' instead).	The backfill policy did not match a defined policy.
0x2100a834	ADMIN	system.moab	ERROR	MWM_NODE_LIST_ALLOCATION	The system was unable to allocate a node list for job '%s' in partition '%s'.	The system might be low on memory.
0x2100a835	ADMIN	system.moab	ERROR	MWM_BAD_NODE_IN_NODELIST	The reservation nodelist for job' %s' has an invalid node at index %s.	Check the nodes specified for the reservation.
0x2100a836	ADMIN	system.moab	ERROR	MWM_RESERVATION_SPANS_PARTITIONS	The reservation request for job '%s' spans partitions (node %s partition %s).	Reservations that span partitions must have the COALLOC flag set.
0x2100a837	ADMIN	system.moab	ERROR	MWM_ADJUST_JOB_RESERVATION_FAILURE	The system failed to adjust job '%s' reservation on node %s.	This is an internal error.
0x2100	ADMIN	system.moab	ER	MWM_OBJECT_	The object type specified (%s) is not valid.	A valid object type must be specified.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
a838			ROR	TYPE_INVALID		
0x2100a839	ADMIN	system.moab	ERROR	MWM_MISSING_OBJECT_ID	The object ID is missing.	A valid object ID must be specified.
0x2100a83a	ADMIN	system.moab	ERROR	MWM_MISSING_ACTION	The action is missing.	A valid action must be specified.
0x2100a83b	ADMIN	system.moab	ERROR	MWM_PIPE_BUFFER_FAILED	The system could not open a bi-directional pipe.	A valid action must be specified.
0x2100a83c	ADMIN	system.moab	ERROR	MWM_STD_OUT_FAILED	Failed to load stdout file '%s'.	Check the file name and path.
0x2100a83d	ADMIN	system.moab	ERROR	MWM_STD_ERR_FAILED	Failed to load stderr file '%s'.	Check the file name and path.
0x2100a841	ADMIN	system.moab	ERROR	MWM_CREATE_NODE_FAILURE	Unable to create node '%s' (check license and MAXNODE parameter).	check license and MAXNODE parameter.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a842	ADMIN	system.moab	ERROR	MWM_PARTITION_CREATE_FAILURE	The system was unable to create a shared partition for the global node.	The system might be low on memory.
0x2100a845	ADMIN	system.moab	ERROR	MWM_HT_FIND_NODE_FAILURE	Cannot find node '%s' in hash table.	A node by the given name might not have been created.
0x2100a847	ADMIN	system.moab	ERROR	MWM_HT_FIND_VM_FAILURE	Cannot find VM '%s' in hash table.	A VM with the given name might not have been created.
0x2100a848	ADMIN	system.moab	ERROR	MWM_COMMAND_FAILED	Command '%s' failed. StatusCode: %s; Response: '%s'.	Check the command syntax and parameters.
0x2100a849	ADMIN	system.moab	ERROR	MWM_HASH_TABLE_INITIALIZATION	There was an unexpected hash table initialization error.	The hash table for jobs to delete never initialized correctly.
0x2100a84e	ADMIN	system.moab	ERROR	MWM_UNABLE_TO_AUTHENTICATE_JOB	Unable to authenticate job %s when UID or GID is empty (UID=%s, GID=%s).	Either the UID or the GID field is empty.
0x2100a84f	ADMIN	system.moab	ERROR	MWM_MISSING_JOB_TASKCOUNT	Job does not have a taskcount specified.	Each job must have an associated taskcount.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a850	ADMIN	system.moab	ERROR	MWM_FAILED_EXCLUDE_NODELIST	The system failed to add an exclude nodelist to a submission.	The job exclude hostlist could not be converted into a string.
0x2100a852	ADMIN	system.moab	ERROR	MWM_CANNOT_SUBMIT_VM_MIGRATION_JOB	Failed to submit migration job for VM %s.	Check MIGRATETEMPLATE on workflow and its trigger.
0x2100a853	ADMIN	system.moab	ERROR	MWM_WEB_SERVICES_WRITE_FAILURE	Error %s encountered while trying to write to web services.	Encountered problem trying to put HTTP data to web server.
0x2100a854	ADMIN	system.moab	ERROR	MWM_WEB_SERVICES_URL_MISSING	Missing URL in call to web services.	Web services must have a valid destination URL.
0x2100a855	ADMIN	system.moab	ERROR	MWM_RM_PARTITION_CREATE_FAILURE	The system was unable to create a partition for RM '%s'.	The system might be low on memory.
0x2100a856	ADMIN	system.moab	ERROR	MWM_PARSE_MPP_NODES_FAILURE	The system failed to parse the MPP nodes value '%s'.	Check the MPP names.
0x2100a857	ADMIN	system.moab	ERR	MWM_FIND_MPP_NODES_FAILURE	The system failed to find node '%s' in the MPP nodes value '%s'.	Check the MPP names.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
			OR			
0x2100a859	ADMIN	system.moab	ERROR	MWM_NODE_SET_TYPE_INVALID	The node set type specified (%s) is not valid.	Check the NODESETLIST option.
0x2100a85a	ADMIN	system.moab	ERROR	MWM_GRES_ADD_FAILURE	Unable to add the GRESTOJOBATTRMAP '%s'.	The limit has been reached.
0x2100a85d	ADMIN	system.moab	ERROR	MWM_NOT_MWS_RM	The resource manager is not Moab Web Services.	Make sure the resource manager has Moab Web Services.
0x2100a85e	ADMIN	system.moab	ERROR	MWM_MWS_RM_CURL_CONNECTION	The system could not initialize a cURL connection to the MWS RM.	The cURL command to connect to the resource manager has failed.
0x2100a85f	ADMIN	system.moab	ERROR	MWM_MWS_RM_CURL_CONNECTION_EXPANDED	Could not connect to MWS RM (%s) at '%s%s' as '%s', response code: %s; cURL error: %s (%s); MWS response: '%s'.	The connection has failed.
0x2100a860	ADMIN	system.moab	ERROR	MWM_MWS_RM_JSON_CLUSTER_QUERY_EMPTY	JSON cluster query data from MWS RM (%s) is null or empty.	The query must contain valid JSON data.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a861	ADMIN	system.moab	ERROR	MWM_MWS_RM_JSON_WORKLOAD_QUERY_EMPTY	JSON workload query data from MWS RM (%s) is null or empty.	The query must contain valid JSON data.
0x2100a867	ADMIN	system.moab	ERROR	MWM_JOB_TRANSITION_FAILURE	Unable to transition a job.	The job was missing requirements.
0x2100a868	ADMIN	system.moab	ERROR	MWM_SET_JOB_VARIABLE	Unable to set a job pref variable.	The system is probably low on memory.
0x2100a869	ADMIN	system.moab	ERROR	MWM_ARRAY_EXPANSION	Unable to expand the size of an array.	The system is probably low on memory.
0x2100a86c	ADMIN	system.moab	ERROR	MWM_VC_FIND_FAILURE	The system could not find the virtual container '%s'.	Check the name of the VC.
0x2100a86d	ADMIN	system.moab	ERROR	MWM_VC_USER_CREDENTIALS	User '%s' does not have access to virtual container '%s'.	Check the rights granted to the VC.
0x2100a86e	ADMIN	system.moab	ERROR	MWM_VC_BEING_DELETED	Virtual container '%s' is being deleted; cannot add jobs to it.	Only add jobs to VCs that are not being deleted.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a86f	ADMIN	system.moab	ERROR	MWM_PARTITION_STATUS	Unable to query the status of a partition - %s.	Check to make sure the resource manager is running.
0x2100a870	ADMIN	system.moab	ERROR	MWM_FIND_JOB_TEMPLATE	The system failed to find job template '%s'.	Check the template name for the given job.
0x2100a873	ADMIN	system.moab	ERROR	MWM_PROCESS_EVENT	Unable to process the generic event.	During processing, unable to get a description of the event.
0x2100a876	ADMIN	system.moab	ERROR	MWM_JOB_CAN_NEVER_RUN	Unable to allocate tasks for job at any time.	Job tasks must match available resources.
0x2100a877	ADMIN	system.moab	ERROR	MWM_NODE_NOT_IN_PARTITION	Node is not associated with any partition.	Node must be in a partition.
0x2100a87a	ADMIN	system.moab	ERROR	MWM_NODE_COUNT_EXCEEDS_LICENSE	The number of nodes '%s' exceeds the current license limit '%s'.	A different license is needed to use more nodes.
0x2100a87b	ADMIN	system.moab	ERROR	MWM_KILL_FAILURE	OS call to kill process (PID: %s) %s failed).	This is an operating system error.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100a87c	ADMIN	system.moab	ERROR	MWM_MISSING_JOB_REQUIREMENTS	Job does not have any requirements specified.	Each job must have requirements attached.
0x2100a87d	ADMIN	system.moab	ERROR	MWM_JOB_MISSING_DISPATCH_TIME	Job loaded in alloc state '%s' with no dispatch time.	The job must have a dispatch time.
0x2100a87e	ADMIN	system.moab	ERROR	MWM_UNEXPECTED_OBJECT_TYPE	The object type '%s' was not expected in this operation.	Verify that a valid object type is given.
0x2100a87f	ADMIN	system.moab	ERROR	MWM_JOB_TRANSITION_XML	Unable to create XML element from job transition object.	The system might be low on memory.
0x2100a881	ADMIN	system.moab	ERROR	MWM_VM_CREATE_RESERVATION	Cannot create reservation for VM '%s'.	Failed to create reservation for the given VM.
0x2100aa08	ADMIN	system.moab	ERROR	MWM_VM_FIELD_VALUE	VM '%s' has an invalid '%s%s%s' field value.	The field value for the VM is invalid.
0x2100aa09	ADMIN	system.moab	ERROR	MWM_NODE_FIELD_VALUE	Node '%s' has an invalid '%s%s%s' field value.	The field value for the node is invalid.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100aa0a	ADMIN	system.moab	ERROR	MWM_JOB_FIELD_VALUE	Job '%s' has an invalid '%s%s%s' field value.	The field value for the job is invalid.
0x2100aa4a	ADMIN	system.moab	ERROR	MWM_INCOMPLETE_JOB_TEMPLATE_DATASTAGING	The job template '%s' has an incomplete data staging system job specification.	Job templates must fully specify the data staging system job.
0x2100aa55	ADMIN	system.moab	ERROR	MWM_CANNOT_ADD_DEFAULT_USER	Cannot add default user.	Default user cannot be added.
0x2100aa56	ADMIN	system.moab	ERROR	MWM_CANNOT_ADD_DEFAULT_ACCOUNT	Cannot add default account.	Default account cannot be added.
0x2100aa57	ADMIN	system.moab	ERROR	MWM_CANNOT_ADD_DEFAULT_CLASS	Cannot add default class.	Default class cannot be added.
0x2100aa58	ADMIN	system.moab	ERROR	MWM_CANNOT_CREATE_CP_FILE	Cannot create checkpoint file. Cannot open %s file '%s', errno: %s (%s).	The fopen() system call failed. Use the errno and associated message to determine possible causes.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x2100aa59	ADMIN	system.moab	ERROR	MWM_PAST_PROC_SEC_LIMIT_DAILY	Maximum daily processor seconds reached - Elastic Computing disabled. Total: %s, Max: %s	The daily threshold has been reached - Elastic Computing disabled.
0x2100aa5a	ADMIN	system.moab	ERROR	MWM_PAST_PROC_SEC_LIMIT_MONTHLY	Maximum monthly processor seconds reached - Elastic Computing disabled. Total: %s, Max: %s	The monthly threshold has been reached - Elastic Computing disabled.
0x2100aa5b	ADMIN	system.moab	ERROR	MWM_PAST_PROC_SEC_LIMIT_QUARTERLY	Maximum quarterly processor seconds reached - Elastic Computing disabled. Total: %s, Max: %s	The quarterly threshold has been reached - Elastic Computing disabled.
0x2100aa5c	ADMIN	system.moab	ERROR	MWM_PAST_PROC_SEC_LIMIT_YEARLY	Maximum yearly processor seconds reached - Elastic Computing disabled. Total: %s, Max: %s	The yearly threshold has been reached - Elastic Computing disabled.
0x2100aa5d	ADMIN	system.moab	ERROR	MWM_PAST_QOS_PROC_SEC_LIMIT_DAILY	Maximum daily processor seconds reached for QOS %s - Elastic Computing disabled. Total: %s, Max: %s	The daily QOS threshold has been reached - Elastic Computing disabled.
0x2100aa5e	ADMIN	system.moab	ERROR	MWM_PAST_QOS_PROC_SEC_LIMIT_MONTHLY	Maximum monthly processor seconds reached for QOS %s - Elastic Computing disabled. Total: %s, Max: %s	The monthly QOS threshold has been reached - Elastic Computing

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						disabled.
0x2100aa5f	ADMIN	system.moab	ERROR	MWM_PAST_QOS_PROC_SEC_LIMIT_QUARTERLY	Maximum quarterly processor seconds reached for QOS %s - Elastic Computing disabled. Total: %s, Max: %s	The quarterly QOS threshold has been reached - Elastic Computing disabled.
0x2100aa60	ADMIN	system.moab	ERROR	MWM_PAST_QOS_PROC_SEC_LIMIT_YEARLY	Maximum yearly processor seconds reached for QOS %s - Elastic Computing disabled. Total: %s, Max: %s	The yearly QOS threshold has been reached - Elastic Computing disabled.
0x2100aa61	ADMIN	system.moab	ERROR	MWM_NODE_CANNOT_LICENSE	Node '%s' does not report socket/gpu/mic information. It cannot be licensed.	A RM that reports socket/gpu/mic information for each node must be used with this license type.
0x2100c001	INTERNAL	system.moab	ERROR	MWM_NOT_IMPLEMENTED	Function %s has not been implemented yet.	This error is used when we've stubbed out code but do not expect it to be called in production environments. It's not helpful except for internal diagnostics.
0x2100	INTER	system.mo	ER	MWM_CANNOT_	Cannot send %s of %s bytes to socket descriptor %s -	The send()

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
e729	NAL	ab	ROR	SEND_TO_ SOCKET_ DETAILED	errno: %s (%s).	system call failed. Socket is blocked (select () indicated socket was available--check MTU).
0x2100e77b	INTERNAL	system.moab	ERROR	MWM_ CLIENT_ COUNT_ NEGATIVE	Client count fell below zero on socket %s.	This is an internal error. The number of client connections should always be zero or greater.
0x2100e784	INTERNAL	system.moab	ERROR	MWM_ HOSTLIST_ MISSING	A hostlist was specified but now it is NULL/EMPTY.	The job claims to have a specified hostlist, but at the current point in processing no list can be found. This is most likely an internal problem.
0x2100e7b4	INTERNAL	system.moab	ERROR	MWM_ REQATTR_ UNSUPPORTED_ OPERATION	Operation (%s) not supported on required attributes (reqattrs).	See documentation for supported operators allows on required attributes (reqattrs).
0x2100	INTER	system.mo	ER	MWM_VM_ NOT_	VM '%s' not linked to VMTracking job '%s' (linked	A VM must be associated with

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
e7ef	NAL	ab	ROR	LINKED_TO_TRACKING_JOB	to job '%s').	a tracking job.
0x31002a2c	USER	system.moab	ALERT	MWM_NO_TASKS_FOUND_ON_JOB	No tasks found for job '%s'.	Check job submission arguments for desired requirements.
0x31008385	ADMIN	system.moab	FATAL	MWM_EXPIRED_LICENSE	%s License has expired.	A license file was found but it has expired. Please contact your sales representative at Adaptive Computing for assistance.
0x31008386	ADMIN	system.moab	FATAL	MWM_EVALUATION_EXPIRED	%s evaluation period has expired.	The evaluation period has expired. Please contact your sales representative at Adaptive Computing for assistance.
0x31008387	ADMIN	system.moab	FATAL	MWM_UNEXPECTED_LICENSE_ERROR	Moab will now exit. Unexpected error while reading license: %s	Moab was unable to verify that the license file was valid. Please contact your sales representative at Adaptive Computing for assistance.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x3100a712	ADMIN	system.moab	FATAL	MWM_UNABLE_TO_ALLOCATE_MEMORY	Unable to allocate memory.	One or more calls to allocate memory failed.
0x3100a714	ADMIN	system.moab	FATAL	MWM_CANNOT_RESTORE_UID	Cannot restore EUID to '%s' for server, errno: %s (%s).	The setuid() system call failed. There was a failure resetting the UID of the process. This might be because the process is running as a different user. Use the errno and associated message to determine possible causes.
0x3100a715	ADMIN	system.moab	FATAL	MWM_CANNOT_CHANGE_UID	Cannot change UID to user '%s' (UID: %s) errno: %s (%s).	The setuid() system call failed. Use the errno and associated message to determine possible causes.
0x3100a716	ADMIN	system.moab	FATAL	MWM_CANNOT_RESTORE_GID	Cannot restore GID to '%s' for server, errno: %s (%s).	The setgid() system call failed. There was a failure resetting the GID of the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						process. This might be because the process is running in a different group. Use the errno and associated message to determine possible causes.
0x3100a717	ADMIN	system.moab	FATAL	MWM_CANNOT_FORK_INTO_BACKGROUND	Cannot fork the process into the background, errno: %s (%s).	The fork() system call failed. Moab must do this to daemonize unless run with the '-d' flag. This is usually due to low system resources.
0x3100a71c	ADMIN	system.moab	FATAL	MWM_CANNOT_CHANGE_OWNERSHIP_FILE_FATAL	Cannot change ownership of %s file to uid:%s gid:%s errno: %s (%s).	The fchown() system call failed. Use the errno and associated message to determine possible causes.
0x3100a745	ADMIN	system.moab	FATAL	MWM_CANNOT_GET_SERVER_HOSTNAME	Cannot determine hostname and attribute '%s' of parameter %s is not specified.	Moab failed to obtain system host name or ip address information from the

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						operating system.
0x3100a746	ADMIN	system.moab	FATAL	MWM_HA_MOAB_NOT_STARTED_ON_CORRECT_HOSTS	The server must be started on host '%s' or on alternate '%s' (currently on '%s').	Moab must be started on either the primary or alternate host for high availability.
0x3100a747	ADMIN	system.moab	FATAL	MWM_MOAB_NOT_STARTED_ON_CORRECT_HOST	The server must be started on host '%s' (currently on '%s').	Moab must be started on specified host as identified by the SCHEDCFG parameter.
0x3100a749	ADMIN	system.moab	FATAL	MWM_MOAB_ALREADY_RUNNING	Moab is already running. Cannot open user interface socket on port %s.	Cannot open user interface socket, which is most likely caused by Moab already running.
0x3100a74b	ADMIN	system.moab	FATAL	MWM_CANNOT_LOCATE_FULL_PATH	Cannot locate the full path for '%s'.	Check the path to make sure the Moab executable is in it. Restart manually to work around this problem temporarily.
0x3100a74c	ADMIN	system.moab	FATAL	MWM_CANNOT_RESTART_SCHEDULER	Exec failed when attempting to restart the scheduler '%s' rc: %s.	Check permissions on this executable to correct and restart

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
						manually to work around.
0x3100a750	ADMIN	system.moab	FATAL	MWM_CANNOT_CONNECT_TO_DB_WITH_STRICT_CONFIG_CHECK_ON	StrictConfigCheck ON and cannot connect to DB--please check DB engine and configuration (%s).	Moab was unable to connect to the database and with strict configuration on Moab must exit.
0x3100a751	ADMIN	system.moab	FATAL	MWM_USER_NOT_AUTHORIZED_TO_RUN_THIS_PROGRAM	The user '%s' (UID: %s) is not authorized to run this program.	The user has insufficient privileges to run the program.
0x3100a752	ADMIN	system.moab	FATAL	MWM_PROBLEMS_WITH_KEY_FILE	Problems with key file.	Key file does not exist or ownership of key file is invalid.
0x3100aa0d	ADMIN	system.moab	FATAL	MWM_STRICT_CHECK_EXIT	Exiting because of strict configuration check.	Moab is configured to exit if there are any errors in configuration files or file/directory layout. One of these errors has occurred.
0x3100c002	INTERNAL	system.moab	FATAL	MWM_TESTING_FATAL	Testing with single argument: %s.	Internal error for testing diagnostics.

Code	Escalation Level	Topic	Severity	Event Name	Message Template	Comment
0x3100e74f	INTERNAL	system.moab	FATAL	MWM_CORRUPT_CHECKPOINT_FILE	Unable to read the checkpoint file.	Please contact Adaptive Computing for assistance.

Appendix D: Adjusting Default Limits

Moab is distributed in a configuration capable of supporting multiple architectures and systems ranging from a few processors to several thousand processors. However, in spite of its flexibility, for performance reasons, it still contains a number of default object limits parameters and static structures defined in header files. These limits constrain such things as the maximum number of jobs, reservations, and nodes that Moab can handle and are set to values that provide a reasonable compromise between capability and memory consumption for most sites. However, many site admins want to increase some of these settings to extend functionality, or decrease them to save consumed memory. The most common parameters are listed in the table below. Parameters listed in the Moab configuration file (`moab.cfg`) can be modified by restarting Moab. To change parameters listed in `moab.h`, contact technical support.

Moab currently possesses hooks to allow sites to create local algorithms for handling site specific needs in several areas. The `contrib` directory contains a number of sample local algorithms for various purposes. The `MLocal.c` module incorporates the algorithm of interest into the main code. The following scheduling areas are currently handled via the `MLocal.c` hooks:

- Local Job Attributes
- Local Node Allocation Policies
- Local Job Priorities
- Local Fairness Policies

CLIENTMAXCONNECTIONS	
Location	`moab.cfg` (dynamic parameter)
Default	128
Max Tested	---
Description	Maximum number of connections that can simultaneously connect to Moab. See the parameter CLIENTMAXCONNECTIONS for additional information.

JOBMAXNODECOUNT	
Location	`moab.cfg` (dynamic parameter)

JOBMAXNODECOUNT	
Default	1024
Max Tested	8192
Description	Maximum number of compute nodes that can be allocated to a job. After changing this parameter, Moab must be restarted for changes to take effect. The value cannot exceed that of the MAXNODE parameter (specified in moab.cfg). If you specify a value higher than the limit set for the `MAXNODE` parameter, the value will match `MAXNODE`. `JOBMAXNODECOUNT` can also be specified within configure using --with-maxjobsize=<NODECOUNT>. See the parameter JOBMAXNODECOUNT for additional information.

JOBMAXTASKCOUNT	
Location	`moab.cfg` (dynamic parameter)
Default	32768
Max Tested	250000
Description	Total number of tasks allowed per job. See the parameter JOBMAXTASKCOUNT for additional information.

MAXGRES	
Location	`moab.cfg` (dynamic parameter)
Default	512
Max Tested	---
Description	Total number of distinct generic resources that can be managed. See the parameter MAXGRES for additional information.

MAXJOB	
Location	`moab.cfg` (dynamic parameter)
Default	51200

MAXJOB	
Max Tested	500,000
Description	Maximum number of jobs that can be evaluated simultaneously. (Can also be specified within **configure** using **--with-maxjobs=<JOBCOUNT>**.) See the parameter MAXJOB for additional information.

MAXNODE	
Location	`moab.cfg` (dynamic parameter)
Default	5120
Max Tested	160000
Description	Maximum number of compute nodes supported. See the parameter MAXNODE for additional information.

MAXRSVPERNODE	
Location	`moab.cfg` (dynamic parameter)
Default	64
Max Tested	1024
Description	Maximum number of reservations a node can simultaneously support. See the parameter MAXRSVPERNODE for additional information.

MMAX_ATTR	
Location	`moab.h`
Default	128
Max Tested	512
Description	Total number of distinct node attributes (PBS node attributes/LL node features) that can be tracked.

MMAX_CLASS	
Location	`moab.h`
Default	24
Max Tested	64
Description	Total number of distinct job classes/queues available.

MMAX_FSDEPTH	
Location	`moab.h`
Default	24
Max Tested	32
Description	Number of active fairshare windows.

MMAX_PAR	
Location	`moab.h`
Default	32
Max Tested	32
Description	Maximum number of partitions supported.

MMAX_QOS	
Location	`moab.h`
Default	128
Max Tested	128
Description	Total number of distinct QoS objects available to jobs.

MMAX_RACK	
Location	`moab.h`
Default	200
Max Tested	200
Description	Total number of distinct rack objects available within cluster.

MMAX_RANGE	
Location	`moab.h`
Default	2048
Max Tested	2048
Description	Total number of distinct timeframes evaluated. This is proportional to the size of the cluster and the number of simultaneously active jobs in the cluster. (Can be specified within `./configure` using `--with-maxrange=<RANGECOUNT>`.) Increasing this value will not increase the size of total memory consumed by Moab but may result in minor slowdowns in the evaluation and optimization of reservations.

MMAX_REQ_PER_JOB	
Location	`moab.h`
Default	5
Max Tested	64
Description	Total number of unique requirement structures a job can have. Limits the number of `-w` clauses in the *`mshow -a`* command. It also limits the number of `-l nodes=X+Y+Z` a normal HPC job can have.

Related Topics

- Appendix I: Considerations for Large Clusters

Appendix E: Security

Moab provides role and host based authorization, encryption, and DES, HMAC, and MD5 based authentication. The following sections describe these features in more detail and tell how to control access to sensitive configuration information.

In this appendix:

E.1 Authentication (Interface Security)

Moab supports password-challenge, DES, HMAC, and MD5 based authentication. Authentication protocols can be specified on a per interface basis allowing independent realms of trust with per realm secret keys and even per realm authentication protocols.

In this section:

E.1.1 Mauth Authentication

Mauth is a tool provided with Moab that provides client authentication services. With Mauth enabled, each client request is packaged with the client ID, a timestamp, and an encrypted key of the entire request generated using the shared secret key.

This tool is enabled by providing a secret key. A random key is selected when the Moab `./configure` script is run and can be regenerated at any time by rerunning `./configure` and rebuilding Moab. If desired, this random key can be overridden by specifying a new key in the protected `.moab.key` file as in the example below.

Moab must be shut down before setting a new secret key. Use the `service moab stop` or *`mschedctl -k`* commands to shut down Moab.

```
> vi /opt/moab/etc/.moab.key
(insert key)
> cat /opt/moab/etc/.moab.key
XXXXXXXX
# secure file by setting owner read-only permissions
> chmod 400 /opt/moab/etc/.moab.key
# verify file is owned by root and permissions allow only root to read file
> ls -l /opt/moab/etc/.moab.key
-r-------- 1 root root 15 2024-04-05 03:47 /opt/moab/etc/.moab.key
```

Be aware of the following:

- All directories in the path containing `.moab.key` must be owned by the root or primary Moab user. It must not be writable by 'other' in its permissions.
- The `.moab.key` file will need to be on each host that is authorized to run Moab client commands.
- The `.moab.key` file must reside in the same directory as the `moab.cfg` file.
- By default:
  - The `.moab.key` file will be owned by the user root and its contents will be read by the Mauth tool, which provides client authorization services. If desired, the ownership of this file can be changed so long as this file is readable by the Moab server and the Mauth tool. This can be accomplished if the Moab primary administrator, the owner of Mauth, and the owner of `.moab.key` are the same.
  - It is up to the individual cluster admins to determine whether to use the `.moab.key` file. For sites with source code, the use of `.moab.key` can be mandated by using `./configure --with-keyfile`.
  - Mauth is located in the install `bin` directory. If an alternative name or alternative file location is desired, this can be specified by setting the `AUTHCMD` attribute of the CLIENTCFG parameter within the `moab.cfg` file as in the following example:

    ```
    CLIENTCFG  AUTHCMD=/opt/sbin/mauth
    ```

E.1.1.A Configuring Peer-Specific Secret Keys

Peer-specific secret keys can be specified using the CLIENTCFG parameter. This key information must be kept secret and consequently can only be specified in the `moab-private.cfg` file. With regard to security, there are two key attributes that can be set.

Other resource managers or clients such as Moab Accounting Manager or a Wiki interface can also use the attributes to configure their authentication algorithms. The default, unless otherwise stated, is always `DES`. These attributes are listed in the table below:

AUTH	
Format	One of `ADMIN1`, `ADMIN2`, or `ADMIN3`
Default	---
Description	The level of control/information available to requests coming from this source/peer.
Example	`CLIENTCFG[RM:clusterB] AUTH=admin1 KEY=14335443`

AUTHTYPE	
Format	One of `DES`, `HMAC`, `HMAC64`, `HMACSHA2`, or `MD5`.
Default	`DES`
Description	The encryption algorithm to use when generating the message checksum.
Example	`CLIENTCFG[AM:mam] AUTHTYPE=HMAC64`

HOST	
Format	`<STRING>`
Default	---
Description	The hostname of the remote peer. Peer requests coming from this host will be authenticated using the specified mechanism. This parameter is optional.
Example	`CLIENTCFG[RM:clusterA] HOST=orx.pb13.com  KEY=banana6`

KEY	
Format	`<STRING>`

KEY	
Default	---
Description	The shared secret key to be used to generate the message checksum.
Example	`CLIENTCFG[RM:clusterA] KEY=banana6`

The `CLIENTCFG` parameter takes a string index indicating which peer service will use the specified attributes. In most cases, this string is simply the defined name of the peer service. However, for the special cases of resource and accounting managers, the peer name should be prepended with the prefix `RM:` or `AM:` respectively, as in `CLIENTCFG [AM:mam]` or `CLIENTCFG[RM:devcluster]`.

The first character of any secret key can be viewed by trusted admins using specific diagnostic commands to analyze Moab interfaces. If needed, increase the length of the secret keys to maintain the desired security level.

E.1.2 MUNGE Authentication

Moab also integrates with MUNGE, an open source authentication service created by Lawrence Livermore National Laboratory (MUNGE Uid 'N' Gid Emporium). MUNGE works with Moab to authenticate user credentials being passed between the Moab client and the Moab server or from Moab server to Moab server.

To set up MUNGE in a cluster or grid, download and install MUNGE on every node in the cluster or grid by following the installation steps at MUNGE Uid 'N' Gid Emporium. The MUNGE secret key must reside on each node in the cluster or grid. Before starting the Moab daemon, the MUNGE daemon must be running on all nodes.

To enable Moab to use MUNGE for authentication purposes, specify the MUNGE executable path in the moab.cfg file using `CLIENTCFG` and `AUTHCMD` as in the following example. The MUNGE executable path must reside in each client's `moab.cfg` file also.

```
CLIENTCFG AUTHCMD=/usr/bin/munge
```

Moab requires that the MUNGE and UNMUNGE executable names be 'munge' and 'unmunge' respectively. It also assumes that the UNMUNGE executable resides in the same directory as the MUNGE executable.

Configuring MUNGE Command Options

Moab also integrates with MUNGE command line options. For example, to set up Moab to use a specific socket that was created when the MUNGE daemon was started, use `CLIENTCFG` and `AUTHCMDOPTIONS` to specify the newly created socket. The `AUTHCMDOPTIONS` attribute, like `AUTHCMD`, must also reside in the client's `moab.cfg` file.

```
CLIENTCFG     AUTHCMD=/usr/bin/munge
CLIENTCFG     AUTHCMDOPTIONS="-S /var/run/munge/munge.socket.2"
```

E.1.3 Server Response Control

If a request is received that is corrupt or cannot be authenticated, Moab will report some limited information to the client indicating the source of the failure, such as 'bad key,' 'malformed header,' and so forth. In the case of highly secure environments, or to minimize the impact of sniffing or denial of service attacks, Moab can be configured to simply drop invalid requests. This is accomplished by adding the `DROPBADREQUEST` attribute to the `CLIENTCFG` parameter in the `moab-private.cfg` file as in the following example:

```
CLIENTCFG[DEFAULT] DROPBADREQUEST=TRUE
```

E.1.4 Checksum Algorithm for Client Authentication

The SERVERCSALGO parameter lets you choose the algorithm used for message digests and message authentication codes:

- HMAC64: the default (SHA-1)
- HMACSHA2: more secure (SHA-512)

If you are using Moab Web Services, then you must set the MWS configuration parameter moab.messageDigestAlgorithm to match the value of SERVERCSALGO. See 'moab.messageDigestAlgorithm' in the *Moab Web Services Administrator Guide* for more information.

E.1.5 Interface Development Notes

Sample checksum generation algorithm code can be found in the Socket Protocol Description document.

E.2 Authorization

Role Based Authorization Security Configuration

Moab provides access control mechanisms to limit how the scheduling environment is managed. The primary means of accomplishing this is through limiting the users and hosts that are trusted and have access to privileged commands and data.

With regard to users, Moab breaks access into three distinct levels:

- **Level 1 Moab Admin (Administrator Access)** - These admins have global access to information and unlimited control over scheduling operations. By default, they are allowed to control scheduler configuration, policies, jobs, reservations, and all scheduling functions. They are also granted access to all available statistics and state information. Level 1 administrators are specified using the ADMINCFG[1] parameter.
- **Level 2 Moab Admin (Operator Access)** - These admins are specified using the ADMINCFG[2] parameter. By default, the users listed under this parameter are allowed to change all job attributes and are granted access to all informational Moab commands.
- **Level 3 Moab Admin (Help Desk Access)** - These admins are specified via the ADMINCFG[3] parameter. By default, they are allowed access to all informational Moab commands. They cannot change scheduler or job attributes.

Configuring Role Based Access

Moab allows site specific tuning of exactly which functions are available to each administrator level. Moab also provides two additional administrator levels (**ADMINCFG[4]** and **ADMINCFG[5]**) that can be used for site specific needs.

ADMINCFG[5] is different from other administrator levels because, when set, all commands are authorized at that level by default. Furthermore, if a service is set at level 5, all other services are disallowed.

To configure Moab role based access, use the ADMINCFG parameter:

```
ADMINCFG[1]    USERS=root,john SERVICES=ALL NAME=admin
ADMINCFG[3]    USERS=joe,mary  SERVICES=mdiag,mrsvctl,mcredctl NAME=power
ADMINCFG[5]    USERS=joy,blake SERVICES=NONE NAME=users
...
```

A NONE in services will still allow users to run showq and checkjob on their own jobs.

To determine the role of system users and what commands they can run, use the mcredctl -q role user:<USERID> command.

Using the SERVICES attribute of the ADMINCFG parameter, access to an arbitrary selection of services can be enabled on a per administrator-level basis. Possible services include the following:

Service	Description
changeparam	Change any scheduling policy or parameter. This command is deprecated. Use mschedctl -m instead
checkjob	View detailed information for any job.
checknode	View detailed information for any node.
mbal	Perform real-time load-balancing of interactive commands.
mcredctl	View and modify credential attributes.
mdiag	Provide diagnostic reports for resources, workload, and scheduling.
mjobctl	Modify, control, and view jobs.
mnodectl	Modify, control, and view nodes.
mrmctl	Modify, control, and view resource managers.
mrsvctl	Modify, control, and view reservations.
mschedctl	Modify, control, and view scheduler behavior.
mshow	View existing configuration and predicted resource availability.
showstats	View all scheduler and credential statistics.
releaseres	Release all reservations. This command is deprecated. Use mrsvctl -r instead.
runjob	Immediately execute any job (see the command mjobctl -x).
setqos	Set QoS on any job. This command is deprecated. Use mjobctl -m instead.

Service	Description
setres	Create any reservation. This command is deprecated. Use mrsvctl -r instead.
setspri	Set system priority on any job. This command is deprecated. Use mjobctl -m instead.
showconfig	Show all scheduler configuration parameters. This command is deprecated. Use mschedctl -l instead.
showres	Show detailed information for any reservation.
showstate	Show detailed information for all jobs, including their locations, and display job error messages, if any.

Account and Class/Queue Admins

While the `ADMINCFG` parameter allows organizations to provide controlled access to scheduling objects, it does not allow for distribution along organizational boundaries. For example, a site may set up a level 3 administrator to be able to view statistics, diagnose jobs, and modify job priorities; it does not provide a way to differentiate one type of job from another. If a site admin wanted to allow control based on the queue or account associated with a job, they would best accomplish this using the credential `MANAGERS` attribute.

A credential manager allows a user to be trusted to administer workload and policies for an associated subgroup of jobs. For example, in the configuration below, a number of queue and account managers are configured:

```
CLASSCFG[orion] MANAGERS=johns
CLASSCFG[xray]  MANAGERS=steve2
CLASSCFG[gamma] MANAGERS=steve2,jpw
ACCOUNTCFG[bio] MANAGERS=charles
```

By default, the specified managers can do anything to a job that the actual job owner could do. By default, this includes the ability to view cumulative and per job statistics, see job details, modify job priorities and holds, cancel and preempt jobs, and otherwise adjust policies and constraints within the associated credential.

E.3 Host Security for Compute Resources

Host level security can vary widely from one site to another with everything from pure on-your-honor based clusters to complete encrypted VLAN based network security and government approved per job scrubbing procedures being used. The following topics describe some best practices in use throughout the industry.

In this topic:

E.3.1 Minimal Host Security Enforcement

For minimal host security, no additional configuration is required.

E.3.2 Medium Host Security Enforcement

- Login Access
  - PAM — Enable/disable access by modifying `/etc/security/access.conf`.
- Processes
  - Kill all processes associated with job user (dedicated).
  - Kill all processes associated with job session (dedicated/shared). Use `ps -ju <USER>` or `ps -js <SESSID>`.
- IPC (Inter-Process Communication)
  - Remove shared memory, semaphores, and message queues (use ipcs/ipcrm).
  - Remove named pipes.
- Network/Global File System Access
  - Explicitly unmount user home and global file systems.
- Local Temporary File Systems
  - Where possible, mount local file systems read-only.
  - Clear `/tmp`, `/scratch` and other publicly available local file systems.

- Remove user files with *shred*; *shred* is a Linux command that first overwrites files completely before removing them, preventing remnant data from surviving on the hard drive.

E.3.3 Strict Host Security Enforcement

- VLAN creation
- Host rebuild
  - U.S. Dept. of Energy Disk/File Sanitization (Clearing, Sanitizing, and Destroying Disks)
  - U.S. Dept. of Defense Scrubbing Software

E.4 Securing Sensitive Configuration Information

The moab.cfg file may include sensitive configuration information, such as user or group fairshare targets that determine job priority and scheduling for individual users or groups. Sensitive configuration information can be moved to a separate file in an access-controlled directory and included in moab.cfg using an #INCLUDE directive. For example, the following commands create a directory that requires root permissions to read or execute, and a .cfg file that can be used for sensitive configuration information:

```
# mkdir -m 500 /opt/moab/etc/secure
# echo "ARRAYJOBPARLOCK TRUE" > /opt/moab/etc/secure/moab.secure.cfg
```

Adding the following line to moab.cfg will cause Moab to use the contents of the protected .cfg file:

```
#INCLUDE secure/moab.secure.cfg
```

Appendix F: Initial Moab Testing

Moab has been designed with a number of key features that allow testing to occur in a *no risk* environment. These features allow you to safely run Moab in test mode even with another scheduler running whether it be an earlier version of Moab or another scheduler altogether. In test mode, Moab collects real-time job and node information from your resource managers and acts as if it were scheduling live. However, its ability to actually affect jobs (that is, start, modify, cancel, charge, and so forth) is disabled.

In this appendix:

F.1 Scheduler Modes

This section describes the test modes Moab offers to provide a minimal configuration for verifying such things as proper configuration and operation.

Central to Moab testing is the `MODE` attribute of the SCHEDCFG parameter. This parameter attribute enables admins to determine how Moab will run. The possible values for `MODE` are `NORMAL`, `MONITOR`, `INTERACTIVE`, `TEST`, `SINGLESTEP`, and `SLAVE`. For example, to request monitor mode operation, include the following in the `moab.cfg` file:

```
SCHEDCFG MODE=MONITOR
```

F.2 Normal Mode

If initial evaluation is complete or not required, you can place the scheduler directly into *production* by setting the `MODE` attribute of the `SCHEDCFG` parameter to `NORMAL` and (re)starting the scheduler.

F.3 Monitor Mode (or Test Mode)

Monitor mode allows evaluation of new Moab releases, configurations, and policies in a risk-free manner. In monitor mode, the scheduler connects to the resource manager(s) and obtains live resource and workload information. Using the policies specified in the `moab.cfg` file, the monitor-mode Moab behaves identical to a live or normal-mode Moab except the ability to start, cancel, or modify jobs is disabled. In addition, allocation management does not occur in monitor mode. This allows safe diagnosis of the scheduling state and behavior using the various diagnostic client commands. Further, the log output can also be evaluated to see if any unexpected situations have arisen. At any point, the scheduler can be dynamically changed from monitor to normal mode to begin *live* scheduling.

To set up Moab in monitor mode, do the following:

```
> vi moab.cfg
   (change the MODE attribute of the SCHEDCFG parameter from NORMAL to MONITOR)
> moab
```

Remember that Moab running in monitor mode will not interfere with your production scheduler.

Running Multiple Moab Instances Simultaneously

If running multiple instances of Moab, whether in simulation, normal, or monitor mode, make certain that each instance resides in a different home directory to prevent conflicts with configuration, log, and statistics files. Before starting each additional Moab, set the `MOABHOMEDIR` environment variable in the execution environment to point to the local home directory. Also, each instance of Moab should run using a different port to avoid conflicts.

If running multiple versions of Moab, not just different Moab modes or configurations, set the `$PATH` variable to point to the appropriate Moab binaries.

To *point* Moab client commands (such as showq) to the proper Moab server, use the appropriate command line arguments or set the environment variable `MOABHOMEDIR` in the client execution environment as in the following example:

```
# point moab clients/server to new configuration
> export MOABHOMEDIR=/opt/moab-monitor
# set path to new binaries (optional)
> export PATH=/opt/moab-monitor/bin:/opt/moab-monitor/sbin:$PATH
# start Moab server
> moab
# query Moab server
> showq
```

moabd is a safe and recommended method of starting Moab if things are not installed in their default locations.

F.4 Interactive Mode

Interactive mode allows for evaluation of new versions and configurations in a manner different from monitor mode. Instead of disabling all resource and job control functions, Moab sends the desired change request to the screen and asks for permission to complete it. For example, before starting a job, Moab will post something like the following to the screen:

```
Command:  start job 1139.ncsa.edu on node list test013,test017,test018,test021
Accept:  (y/n) [default: n]?
```

The admin must specifically accept each command request after verifying that it correctly meets desired site policies. Moab will then execute the specified command. This mode is useful in validating scheduler behavior and can be used until configuration is appropriately tuned and all parties are comfortable with the scheduler's performance. In most cases, sites will want to set the scheduling mode to normal after verifying correct behavior.

Appendix G: Integrating Other Resources with Moab

Moab can interface with most popular resource managers, many cluster services, and numerous general protocols. The following topics provide additional information.

In this appendix:

G.1 Compute Resource Managers

G.1.1 Moab-Torque Integration Guide

In this topic:

G.1.1.A Integration Steps

Install Torque

- Installing Torque Resource Manager

Keep track of the PBS target directory, `$PBSTARGDIR`

Install Moab

- Untar the Moab distribution file.
- Change the directory to the `moab-10.1.0` directory.
- Run `./configure`.
- Specify the PBS target directory (**$PBSTARGDIR**) when queried by `./configure`.

Moab interfaces to PBS by utilizing a few PBS libraries and include files. If you have a non-standard PBS installation, you may need to modify `Makefile` and change `PBSIP` and `PBSLP` values and references as necessary for your local site configuration.

The `./configure` script automatically sets up Moab so that the user running configure will become the default Primary Moab Admin (`$MOABADMIN`). This can be changed by modifying the `ADMINCFG[1] USERS=<USERNAME>` line in the Moab configuration file (`moab.cfg`). The primary administrator is the first user listed in the `USERS` attribute and is the ID under which the Moab daemon runs.

Some Tru64 and IRIX systems have a local `libnet` library that conflicts with PBS's `libnet` library. To resolve this, try setting `PBSLIB` to '`${PBSLIBDIR}/libnet.a -lpbs`' in the Moab `Makefile`.

Moab is 64-bit compatible. If PBS/Torque is running in 64-bit mode, Moab likewise needs to be built in this manner to use the PBS scheduling API (i.e., for IRIX compilers, add `-64` to `OSCCFLAGS` and `OSLDFLAGS` variables in the `Makefile`).

When starting both Torque and Moab, it is best to have a small delay between starting the servers. In general (and especially for very fast or very large systems) this is the recommended startup procedure:

- Start Torque.
- Start Moab with scheduling paused (moab -P) to give it a chance to load everything in the checkpoint file and to sync with Torque.
- Unpause Moab with mschedctl -r.

Configure Torque

General Configuration for All Versions of Torque

- Make $MOABADMIN a PBS admin.
    - By default, Moab only communicates with the *`pbs_server`* daemons and the `$MOABADMIN` should be authorized to talk to this daemon.
- *(OPTIONAL)* Set default PBS queue, nodecount, and walltime attributes.
- *(OPTIONAL - Torque only)* Configure Torque to report completed job information by

setting the *qmgr* `keep_completed` parameter:

```
> qmgr -c 'set server keep_completed = 300'
```

PBS nodes can be configured as *time shared* or *space shared* according to local needs. In almost all cases, space shared nodes provide the desired behavior.

PBS/Torque supports the concept of virtual nodes. Using this feature, Moab can individually schedule processors on SMP nodes. See the *Torque Resource Manager Administrator Guide* for information on how to set up the `$PBS_HOME/server_priv/nodes` file to enable this capability. (For example, `<NODENAME>np=<VIRTUAL NODE COUNT>`).

Version-Specific Configuration for Torque

Do not start the *pbs_sched* daemon. This is the default scheduler for Torque; Moab provides this service.

Moab uses PBS's scheduling port to obtain real-time event information from PBS regarding job and node transitions. Leaving the default *qmgr* setting of `set server scheduling=True` allows Moab to receive and process this real-time information.

Configure Moab

By default, Moab automatically interfaces with Torque/PBS when it is installed. Consequently, in most cases, the following steps are not required:

- Specify PBS as the primary resource manager by setting `RMCFG[base] TYPE=PBS` in the Moab configuration file (`moab.cfg`).

If a non-standard PBS installation/configuration is being used, additional Moab parameters may be required to enable the Moab/PBS interface as in the line `RMCFG[base] HOST=$PBSSERVERHOSTPORT=$PBSSERVERPORT`. See 11.1 Resource Manager for more information.

Moab's user interface port is set using the SCHEDCFG parameter and is used for user-scheduler communication. This port must be different from the PBS scheduler port used for resource manager-scheduler communication.

G.1.1.B Torque/Moab Considerations

The default meaning of a node for Torque and Moab are not the same. By default, a node is a host in Torque. The node can have one or more execution slots (procs) allocated to it in the TORQUE_HOME/server_priv/nodes file. However, the number of nodes recognized by Torque is equivalent to the number of node entries in the `TORQUE_HOME/server_priv/nodes` file. A node specification from *qsub* such as `-l nodes=2:ppn=2` will direct Torque to allocate to execution slots on two separate nodes.

Moab is more liberal in its interpretations of a node. To Moab, the qsub request above is interpreted to mean allocate four tasks with at least two tasks on a node. Where Torque would require two nodes for the request, Moab will place all four tasks on the name node (host) if four execution slots are available.

If a cluster has four nodes with eight processors each, Torque still sees only four nodes. Moab sees 32 nodes. However, if a user made a *qsub* request with `-l nodes=10`, Torque would reject the request because there are only four nodes available. To enable Torque to accommodate Moab's more liberal node interpretation, the server parameter available_resources.nodect can be set as a server parameter in Torque. The value of `available_resources.nodect` should equal at least the number of execution slots in the cluster.

For our example, cluster `available_resources.nodect` should be `32`. With this parameter set, the user can now make a request such as `-l nodes=8:ppn=2`. In this example, the user is still limited to a maximum node request of 32.

With `available_resources.nodect` set in Torque, Moab can be directed to honor the default Torque behavior by setting the parameter JOBNODEMATCHPOLICY to `EXACTNODE`.

PBS Features Not Supported by Moab

Moab supports basic scheduling of all PBS node specifications.

Moab Features Not Supported by PBS

PBS does not support the concept of a job QoS or other extended scheduling features by default.

G.1.1.C Troubleshooting

On TRU64 systems, the PBS `libpbs` library does not properly export a number of symbols required by Moab. This can be worked around by modifying the Moab `Makefile` to link the PBS `rm.o` object file directly into Moab.

G.2 Hardware Integration

G.2.1 Moab-NUMA-Support Integration Guide

This section is for NUMA-support systems on large-scale SLES systems using SGI Altix and UV hardware only and requires Torque 3.0 or later.

Scheduling a shared-memory NUMA type system (not the same as a modern SMP-based individual compute node, which cannot share memory between compute nodes) requires some special configuration. Additionally, Moab can use NODESETs to guarantee feasibility of large memory jobs and to enforce node allocation based on the system's interconnect network topology.

In this topic:

G.2.1.A Configuration

To Integrate Moab and NUMA

1. Configure Moab to schedule large memory jobs. Because Moab creates a partition for each resource manager by default, you must configure the cluster controlled by the resource manager to be a shared-memory system to support jobs spanning multiple nodes/blades. To do so, use the PARCFG parameter:

```
RMCFG[sys-uv]  TYPE=Torque
PARCFG[sys-uv] FLAGS=SharedMem
```

   Cluster `sys-uv` is now configured as a shared-memory system to Moab.

2. Configure `NODESET`s as shown below:

```
NODESETISOPTIONAL FALSE
NODESETATTRIBUTE FEATURE
NODESETPOLICY ONEOF
NODESETPRIORITYTYPE FIRSTFIT
```

   The `NODESET` parameters tell Moab that performing node allocation using node sets is required, that the node set name is a feature name assigned to compute nodes, that a job must fit within the available nodes of one node set, and that Moab should use the first node set that contains sufficient available nodes to satisfy the job's request.

3. To configure Moab to perform topology-aware node allocation using node sets, you must create a node set definition for each set of nodes that has the same number of maximum network 'hops' from any node to every other node within the node set. For an example, see the following sample scenario:

   Use Case

   The SGI UV 1000 has a two-socket blade with a physical organization of 16 blades within a blade chassis (SGI term is Intra-Rack Unit or IRU), two blade chassis (IRUs) within a rack, and up to four racks within a single UV system. The UV 1000 interconnect network has a topology that requires zero hops between the two sockets on the same physical blade, one hop between an even-odd blade pair (e.g., blades 0 and 1, 2 and 3, etc.), two hops between all even-numbered or all odd-numbered blades within an IRU, three hops maximum between all blades within an IRU, four hops maximum between all even-numbered blades or all odd-numbered blades within a UV system, and five hops maximum between all blades within a UV system.

   a. Define topology-aware node definitions to parallel the compute nodes reachable within a specific hop count. For the UV 1000, this means the sockets of each blade will belong to six separate node set definitions (i.e., one each for 0, 1, 2, 3, 4, and 5 hops).

   b. Define multiple node sets for different nodes reachable in a specific hop count based on the context of where they are in the network topology; that is, you must create a separate and distinct node set definition for each pair of blades reachable with one hop, for each IRU for its nodes reachable in three hops, etc.

   c. Moab node sets are usually defined as compute node features; that is, each node set defined to Moab should appear as a 'feature' name on one or more compute nodes. Which node set/feature names appear on each compute node depends on where the compute node is in the interconnect network topology.

      Since the SGI UV operating system identifies each blade socket as a separate NUMA node, each NUMA node within a UV system is traditionally an individual compute node to Moab (although Torque has the ability to redefine a compute node definition by grouping OS NUMA nodes, which some UV installations do to define a blade as a compute node).

      For the sake of illustration, this example assumes each OS NUMA node, which is a UV blade socket, is also a compute node in Moab. This means each compute node (blade socket) will have six feature names assigned, where each feature name must reflect both the compute node's location in the network topology and the hop count the name represents. A feature name is constructed by using the same root name for a hop count and a number for the topology location at the hop-count level.

For example, the root feature name 'blade' represents the zero-hop count and the numbers '0', '1', etc, represent the consecutively numbered blades throughout the entire UV system, which yields feature names of 'blade0' for the first blade in the system, 'blade1' for the second blade, etc, to 'blade127' for the last blade in a fully populated 4-rack UV system. To illustrate further, the root feature name 'iru' represents the 3-hops count and the numbers '0' through '7' represent the eight IRUs within a full 4-rack UV system.

d. For each compute node, configure the correct feature name for each of the hop counts possible and its location within the topology at the hop-count level (e.g., blade (0 hops), blade pair (1 hop), odd- or even-numbered nodes within an IRU (2 hops), IRU (3 hops), odd- or even-numbered nodes within the UV (4 hops), and UV system (5 hops)). The following example illustrates the feature names assigned to the compute nodes for an SGI UV 1000 system using the following root feature names:

- blade (0 hops)
- pair (1 hop)
- eiru (2 hops for even-numbered blades within an IRU)
- oiru (2 hops for odd-numbered blades within an IRU)
- iru (3 hops)
- esys (4 hops for even-numbered blades within a UV system)
- osys (4 hops for odd-numbered blades within a UV system)
- sys (5 hops)

Note that nodes 0 and 1 are not given any feature names. This is because the operating system instance for the UV system runs on the first blade and in order to not adversely affect OS performance, no jobs should run on the same compute resources as the operating system; therefore, these nodes have no node set feature names and therefore will never be chosen to run jobs. In addition, some of the first feature names at a specific hop count-level are omitted (such as pair0) since it makes no sense to define them when the first blade is a substantial part of the nodes making up a node set.

The node name of a UV system has the same name as the UV system's host name plus the NUMA node's relative socket number.

```
/var/spool/torque/server_priv/nodes:
sys-uv0
sys-uv1
sys-uv2   blade1          oiru0 iru0 osys sys
sys-uv3   blade1          oiru0 iru0 osys sys
sys-uv4   blade2   pair1  eiru0 iru0 esys sys
sys-uv5   blade2   pair1  eiru0 iru0 esys sys
sys-uv6   blade3   pair1  oiru0 iru0 osys sys
sys-uv7   blade3   pair1  oiru0 iru0 osys sys
```

```
sys-uv8   blade4   pair2  eiru0 iru0 esys sys
sys-uv9   blade4   pair2  eiru0 iru0 esys sys
sys-uv10  blade5   pair2  oiru0 iru0 osys sys
sys-uv11  blade5   pair2  oiru0 iru0 osys sys
sys-uv12  blade6   pair3  eiru0 iru0 esys sys
sys-uv13  blade6   pair3  eiru0 iru0 esys sys
sys-uv14  blade7   pair3  oiru0 iru0 osys sys
sys-uv15  blade7   pair3  oiru0 iru0 osys sys
sys-uv16  blade8   pair4  eiru0 iru0 esys sys
sys-uv17  blade8   pair4  eiru0 iru0 esys sys
sys-uv18  blade9   pair4  oiru0 iru0 osys sys
sys-uv19  blade9   pair4  oiru0 iru0 osys sys
sys-uv20  blade10  pair5  eiru0 iru0 esys sys
sys-uv21  blade10  pair5  eiru0 iru0 esys sys
sys-uv22  blade11  pair5  oiru0 iru0 osys sys
sys-uv23  blade11  pair5  oiru0 iru0 osys sys
sys-uv24  blade12  pair6  eiru0 iru0 esys sys
sys-uv25  blade12  pair6  eiru0 iru0 esys sys
sys-uv26  blade13  pair6  oiru0 iru0 osys sys
sys-uv27  blade13  pair6  oiru0 iru0 osys sys
sys-uv28  blade14  pair7  eiru0 iru0 esys sys
sys-uv29  blade14  pair7  eiru0 iru0 esys sys
sys-uv30  blade15  pair7  oiru0 iru0 osys sys
sys-uv31  blade15  pair7  oiru0 iru0 osys sys
sys-uv32  blade16  pair8  eiru1 iru1 esys sys
sys-uv33  blade16  pair8  eiru1 iru1 esys sys
sys-uv34  blade17  pair9  oiru1 iru1 osys sys
sys-uv35  blade17  pair9  oiru1 iru1 osys sys
...
sys-uv62  blade31  pair15 oiru1 iru1 osys sys
sys-uv63  blade31  pair15 oiru1 iru1 osys sys
sys-uv64  blade32  pair16 eiru2 iru2 esys sys
sys-uv65  blade32  pair16 eiru2 iru2 esys sys
...
sys-uv126 blade63  pair31 oiru3 iru3 osys sys
sys-uv127 blade63  pair31 oiru3 iru3 osys sys
sys-uv128 blade64  pair32 eiru4 iru4 esys sys
sys-uv129 blade64  pair32 eiru4 iru4 esys sys
...
sys-uv190 blade95  pair47 oiru5 iru5 osys sys
sys-uv191 blade95  pair47 oiru5 iru5 osys sys
sys-uv192 blade96  pair48 eiru6 iru6 esys sys
sys-uv193 blade96  pair48 eiru6 iru6 esys sys
...
sys-uv252 blade126 pair63 eiru7 iru7 esys sys
sys-uv253 blade126 pair63 eiru7 iru7 esys sys
sys-uv254 blade127 pair63 oiru7 iru7 osys sys
sys-uv255 blade127 pair63 oiru7 iru7 osys sys
```

4. Define the order that Moab should check node sets for available nodes. Since the parameter NODESETPRIORITYTYPE has a value of `FIRSTFIT`, the node sets must be ordered from smallest to largest so Moab will always choose the node set with the fewest nodes required to satisfy the job's request. This means listing all blades, blade pairs, even and odd IRUs, IRUs, even and odd system, and system, respectively.

```
moab.cfg:
NODESETLIST
blade1,blade2,blade3,…,blade127,pair1,pair2,pair3,…,pair63,eiru0,oiru0,eiru1,oiru1,
…,eiru7,oiru7,iru0,iru1,…,iru7,esys,osys,sys
```

5. Configure Moab to use the `PRIORITYNODEALLOCATIONPOLICY`. This allocation policy causes Moab to allocate enough nodes to fulfill a job's processor and memory requirement.

   ```
   NODEALLOCATIONPOLICY PRIORITY
   ```

6. Set `NODEACCESSPOLICY` to `SINGLEJOB` to ensure that Moab will schedule large memory requests correctly and efficiently. This is necessary even when a job uses only the memory of a NUMA node.

   ```
   NODEACCESSPOLICY SINGLEJOB
   ```

   The policy `SINGLEJOB` tells Moab not to allow jobs to share NUMA resources (cores and memory), which for a shared-memory system is very important for fast job execution. For example, if Moab scheduled a job to use the cores of a NUMA node where memory is used by another job, both jobs would execute slowly (up to 10 times more slowly).

G.2.1.B Job Submission

Jobs can request processors and memory using the `-l nodes=<number of cpus>` and `-l mem=<amount of memory>` syntaxes. You should not have `JOBNODEMATCHPOLICYEXACTNODE` configured on a NUMA system. You must use the `sharedmem` job flag on submission to force the job to run only on a sharedmem partition or cluster and to indicate that the job can span multiple nodes. For example:

```
qsub -l nodes=3,mem=640sgb,flags=sharedmem
```

G.3 Torque/PBS Integration Guide - RM Access Control

In this topic:

G.3.1 Server Configuration

Using the PBS qmgr command, add the Moab Admin as both a *manager* and *operator*:

```
> qmgr
Qmgr: set server managers += <MOABADMIN>@*.<YOURDOMAIN>
```

```
Qmgr: set server operators += <MOABADMIN>@*.<YOURDOMAIN>
Qmgr: quit
```

For example:

```
> qmgr
Qmgr: set server managers += staff@*.ucsd.edu
Qmgr: set operators += staff@*.ucsd.edu
Qmgr: quit
```

If desired, the Moab Admin can be enabled as a manager and operator only on the host on which Moab is running by replacing '*.<YOURDOMAIN>' with '<MOABSERVERHOSTNAME>'.

G.3.2 (Optional) MOM Configuration

If direct Moab to *`pbs_mom`* communication is required, the `mom_priv/config` file on each compute node where *`pbs_mom`* runs should be set as in the following example:

```
$restricted *.<YOURDOMAIN>
$clienthost <MOABSERVERHOSTNAME>
```

For security purposes, sites may want to run Moab under a non-root user ID. If so, and Moab-pbs_mom communication is required, the `mom_priv/config` files must be world-readable and contain the line '`$restricted *.<YOURDOMAIN>`'. (i.e., '`$restricted *.uconn.edu`').

G.4 Torque/PBS Config - Default Queue Settings

In this section:

G.4.1 Default Queue

To set the default queue (the queue used by jobs if a queue is not explicitly specified by the user), set the following:

```
>> qmgr
Qmgr: set system default_queue = <QUEUENAME>
Qmgr: quit
```

G.4.2 Queue Default Node and Walltime Attributes

To set a default of one node and 15 minutes of walltime for a particular queue, set the following:

```
> qmgr
Qmgr: set queue <QUEUENAME> resources_default.nodect = 1
Qmgr: set queue <QUEUENAME> resources_default.walltime = 00:15:00
Qmgr: quit
```

G.4.3 System-Wide Default Node and Walltime Attributes

To set system wide defaults, set the following:

```
> qmgr
Qmgr: set server resources_default.nodect = 1
Qmgr: set server resources_default.walltime = 00:15:00
Qmgr: quit
```

G.5 Provisioning Resource Managers

G.5.1 Validating an xCAT Installation for Use with Moab

In this topic:

G.5.1.A Introduction to Validating xCAT Configuration

This document describes a series of steps to validate xCAT configuration prior to configuring Moab to manage hardware via xCAT. It is assumed the reader is familiar with xCAT and the xCAT configuration on the target site. This guide does not provide xCAT configuration documentation or troubleshooting information; refer to the xCAT documentation for such information.

G.5.1.B Verifying Node List

Verify that all nodes that Moab will manage are known to xCAT with the xCAT *nodels* command. Ensure that all expected (and no unexpected) nodes are listed. You may find it useful to create new group names to identify Moab-managed nodes.

```
[root@h0 moab]# nodels hyper,compute
h1
h2
h3
h4
h5
h7
kvmm1
kvmm10
kvmm2
kvmm3
kvmm4
kvmm5
kvmm6
kvmm7
kvmm8
[root@h0 moab]#
```

G.5.1.C Reporting Node Status

Verify that all nodes report their status correctly using the xCAT *nodestat* command. Ensure that all nodes show the correct status (`sshd`, `installing`, `noping`, and so forth); there should not be any timeouts or error messages.

```
[root@h0 moab]# nodestat hyper,compute |sort
h1: pbs,sshd
h2: pbs,sshd
h3: pbs,sshd
h4: pbs,sshd
h5: pbs,sshd
h7: noping
kvmm10: noping
kvmm1: pbs,sshd
kvmm2: pbs,sshd
kvmm3: pbs,sshd
kvmm4: pbs,sshd
kvmm5: pbs,sshd
kvmm6: pbs,sshd
kvmm7: pbs,sshd
kvmm8: noping
kvmm9: noping
[root@h0 moab]#
```

G.5.1.D Verifying Hardware Management Configuration

Verify that all nodes that Moab will manage have hardware management interfaces correctly configured using the xCAT *nodels* and *rpower* commands. After each of the *rpower* commands, verify the requested state was achieved with *rpower stat*.

```
[root@h0 moab]# nodels h1,kvmm1 nodehm.mgt nodehm.power
h1: nodehm.power: ilo
h1: nodehm.mgt: ilo
kvmm1: nodehm.power: kvm
kvmm1: nodehm.mgt: kvm
[root@h0 moab]# rpower h1,kvmm1 off
h1: off
kvmm1: off
[root@h0 moab]# rpower h1,kvmm1 stat
h1: off
kvmm1: off
[root@h0 moab]# rpower h1,kvmm1 boot
h1: on reset
kvmm1: on reset
[root@h0 moab]# rpower h1,kvmm1 stat
h1: on
kvmm1: on
[root@h0 moab]#
```

G.5.1.E Verifying Provisioning Images

Verify that all operating system images that Moab uses are configured correctly in xCAT. For stateful images, test that all combinations of operating system, architecture, and profile install correctly.

```
[root@h0 moab]# rinstall -o centosX.X -a x86_64 -p hyper h1
h1: install centosX.X-x86_64-hyper
h1: on reset
[root@n100 ~]# sleep 15 && nodestat n05
n05: ping install centosX.X-x86_64-hyper
[root@h0 moab]#
```

For stateless images, test that nodes are able to network boot the images:

```
[root@h0 moab]# nodech h5 nodetype.os=centosX.X nodetype.arch=x86_64
nodetype.profile=hyper
[root@h0 moab]# nodeset h5 netboot
h5: netboot centosX.X-x86_64-hyper
[root@h0 moab]# rpower h5 boot
h5: on reset
[root@h0 moab]# sleep 60 && nodestat h5
h5: pbs, sshd
[root@h0 moab]#
```

Related Topics

- 11.5.1 Native Resource Manager Interface Overview
- 11.8 Resource Provisioning

Appendix H: Interfacing with Moab (APIs)

Moab provides numerous interfaces allowing it to monitor and manage most services and resources. It also possesses flexible interfaces to allow it to interact with peer services and applications as both a broker and an information service. This appendix is designed to provide a general overview and links to more detailed interface documentation.

Moab interfaces to systems providing various services and using various protocols. This appendix is designed to assist users who want to enable Moab in new environments using one of the existing interfaces. It does not cover the steps required to create a new interface.

In this appendix:

H.1 Accounting Interfaces

Moab accounting interfaces allow Moab to export local utilization statistics, events, and accounting information to site specific scripts.

H.2 Grid Interfaces

Moab provides interfaces to allow interaction with various grid brokers and services. These interfaces allow Moab to provide services and utilize services.

In this section:

H.2.1 Services Utilized

- Information Services (import and utilize information service data in making scheduling decisions)
- Job Migration
- Data Migration
- Credential Mapping
- Security and Delegation

See Chapter 23: Moab Workload Manager for Grids for more information on utilized services.

H.2.2 Services Provided

- Information Services (provide resource, workload, and credential information)
- Job Migration
- Data Migration
- Credential Mapping

See 23.1 Grid Basics for more information on provided services.

H.3 Identity and Credential Management Interfaces

Moab's identity and credential management interfaces allow Moab to exchange credential and user configuration, access, policy, and usage information:

- 18.4 Identity Managers
- 5.5.1 Accounting Manager
- Chapter 23: Moab Workload Manager for Grids

H.4 Job Submission and Management Interface

Moab's Job Submission and Management Interface provides interfaces to query resource availability, submit, modify, and manage jobs, and query the status of active and completed jobs:

- Resource Availability Query:
  - Determine quantity, state, and configuration of configured resources (idle, busy, and down nodes)
  - Determine quantity and configuration of all available resources (idle nodes)
  - Determine resources available subject now and in the future for potential job
  - Determine best target cluster destination for potential job
  - Determine largest/longest job that could start immediately
  - Determine estimated start time for potential job
  - Determine earliest guaranteed start time for potential job
- Reserve Resources:
  - Reserve specific resources for desired time frame
- Submit Job (XML format):
  - Submit job to specific cluster
  - Submit job to global job queue
- Manage Job:
  - Hold job
  - Adjust job priority
  - Modify job executable, args, data requirements, job dependencies, duration, hostcount, or other attributes
  - Suspend/resume job
  - Checkpoint/requeue job
  - Cancel job
  - Migrate job
  - Adjust job quality of service (QoS)
- Query Job:
  - Determine job state, utilization, or output results for idle, active, or completed job
  - Determine estimated start time
  - Determine guaranteed start time

H.5 Query and Control APIs

The Moab Cluster and Grid Suites provides a (Moab) workload manager server that supports a broad array of client services. These services can be directly accessed via Moab client commands. The Query and Control APIs allow external portals and services to obtain information about compute resources, workload, and usage statistics.

CLI (Command Line Interface) XML API

All Moab client commands can report results in XML format to allow the information to be easily integrated into peer services, portals, databases, and other applications. To request that a client command report its output in XML, specify the `--format=xml` flag as in the following example:

```
> showq --format=xml
<Data>
<Object>queue</Object>
<cluster LocalActiveNodes="1" LocalAllocProcs="1" LocalIdleNodes="0"
LocalIdleProcs="3" LocalUpNodes="1"
  LocalUpProcs="4" RemoteActiveNodes="0" RemoteAllocProcs="0" RemoteIdleNodes="0"
RemoteIdleProcs="0"
  RemoteUpNodes="0" RemoteUpProcs="0" time="1128451812"></cluster>
<queue count="1" option="active">
<job AWDuration="11672" EEDuration="1128451812" Group="[DEFAULT]" JobID="Moab.2"
MasterHost="cw2" PAL="2"
  QOS="bug3" ReqAWDuration="54000" ReqNodes="1" ReqProcs="1" RsvStartTime="1128451812"
RunPriority="0"
  StartPriority="1" StartTime="1128451812" StatPSDed="11886.580000"
StatPSUtl="11886.580000" State="Running"
  SubmissionTime="1128451812" SuspendDuration="0" User="smith"></job>
</queue>
<queue count="1" option="eligible">
<job EEDuration="1128451812" Group="jacksond" JobID="customer.35" QOS="bug"
ReqAWDuration="3600"
  ReqProcs="1" StartPriority="1" StartTime="0" State="Idle"
SubmissionTime="1128451812" SuspendDuration="0"
  User="johnson"></job>
<queue><queue count="0" option="blocked"></queue>
</Data>
```

Common Query/Control Services

- jobs
  - query status - mdiag -j (XML details)
  - submit - msub (XML format)
  - cancel - mjobctl -c

- nodes
    - query status - mdiag -n (XML details)
    - create resource reservation - mrsvctl -c
    - destroy resource reservation - mrsvctl -r

H.6 Resource Management Interfaces

Moab can monitor, schedule, and control services and resources using multiple protocols. These protocols include the following:

- script/flat file
- Resource Manager Specific Interfaces - Torque

Using the resource manager interfaces, Moab can do the following:

- monitor resources (compute host, network, storage, and software license based resources):
    - load configuration, architecture, and feature information
    - load state, utilization, and workload information
    - load policy and ownership information
- manage resources:
    - dynamically reconfigure and reprovision resource hardware (processors, memory, etc.):
    - dynamically reconfigure and reprovision resource software (operating system, application software, filesystem mounts, etc.)
    - dynamically reconfigure and reprovision resource security (VPNs, VLANs, host security, etc.)
- monitor workload (batch jobs, interactive jobs, persistent services, dynamic services, distributed services):
    - load state, resource requirement, and required environment information
    - load user, group, and credential information
    - load utilization, resource allocation, and policy information

- manage workload:
  - migrate jobs from one resource to another (intra-cluster and inter-cluster)
  - modify jobs for translation and optimization purposes
  - suspend, resume, checkpoint, restart, and cancel jobs
- query cluster policies and configuration

Appendix I: Considerations for Large Clusters

There are several key considerations in getting a batch system to scale.

In this appendix:

Related Topics

- Appendix D: Adjusting Default Limits

I.1 Handling Large Jobs

For large jobs, additional parameters beyond those specified for large node systems might be required. These include settings for the maximum number of tasks per job, and the maximum number of nodes per job.

I.2 Handling Large Numbers of Jobs

In this section:

I.2.1 Set a Minimum RMPOLLINTERVAL

With event driven resource managers like Torque, each time a job is submitted the resource manager notifies the scheduler. In an attempt to minimize response time, the scheduler starts a new scheduling cycle to determine if the newly submitted job can run. In systems with large numbers of jobs submitted at once, this might not result in the desired behavior for two reasons. First, by scheduling at every job submission Moab schedules newly submitted jobs onto available resources in a first come, first served basis rather than evaluating the entire group of new jobs at once and optimizing the placement accordingly. Second, by launching a scheduling iteration for every job submitted, Moab places a heavy load on the resource manager. For example, if a user were to submit 1000 new jobs simultaneously, for each job submitted, the resource manager contacts the scheduler, the scheduler starts a new iteration, and in this iteration, the scheduler contacts the resource manager requesting updated information on all jobs and resources available.

Setting a minimum `RMPOLLINTERVAL` causes the scheduler to not process jobs as quickly as they are submitted, but rather to wait a minimum amount of time to allow more jobs be submitted and to process these new jobs in groups:

```
RMPOLLINTERVAL 30,60
```

If the system is busy, schedule every 30 seconds. If it is not busy, schedule every 60 seconds.

I.2.2 Reduce Command Processing Time

If your system's scheduling cycle regularly takes longer than the `CLIENTTIMEOUT` value, you can configure Moab to fork a copy of itself that will respond to certain information-only client commands (checkjob, showbf, showres, and showstart). This enables you to run intense diagnostic commands while Moab is in the middle of its scheduling process.

When you set `UIMANAGEMENTPOLICYFORKCLIENTUIPORT<port number>` on the server side, Moab forks a copy of itself that will listen for client commands on a separate port. For example, systems that run client commands, such as submit hosts, can set `CLIENTUIPORT41560`. This will allow the clients to run commands such as *`checkjob`*, `showbf`, `showres` and `showstart` and have cached information returned from the previous scheduling iteration. Moab prints a disclaimer at the top of each command that was populated by the forked process stating that the information may be an iteration behind.

See CLIENTTIMEOUT, CLIENTUIPORT, and UIMANAGEMENTPOLICY for parameter information.

Example I-1: Sample configuration:

```
UIMANAGEMENTPOLICY      FORK
CLIENTUIPORT            41560
```

Moab forks a copy of itself on port 41560, where it will watch for checkjob, showbf, showres, and showstart commands until the main scheduling process completes.

Example I-2: Sample command output:

```
$ checkjob 34

--------------------------------------------------------------------
NOTE: The following information has been cached by the remote server
and may be slightly out of date.
--------------------------------------------------------------------

job 34

State: Idle
Creds: user:wightman group:company class:batch
WallTime: 00:00:00 of 00:01:00
SubmitTime: Thu May 22 14:17:06
(Time Queued Total: 00:00:18 Eligible: 00:00:18)

TemplateSets: DEFAULT
Total Requested Tasks: 1

Req[0] TaskCount: 1 Partition: ALL

SystemID: scale
SystemJID: 34

IWD: $HOME/test/scale
SubmitDir: $HOME/test/scale
Executable: sleep 60
```

I.2.3 Minimize Job Processing Time

Use the ENABLEHIGHTHROUGHPUT parameter. By default, Moab processes all job attributes, filters, remap classes, job arrays, and other information when a job is submitted. This requires full access to the Moab configuration and significantly increases the processing time Moab needs when jobs are submitted. By setting

ENABLEHIGHTHROUGHPUT to TRUE, Moab stores the job information in an internal queue and returns the job ID immediately. The internal queue is processed when Moab begins its next scheduling iteration. This enables Moab to process hundreds of jobs per second rather than 20-30 per second. Because the jobs are processed in a separate queue after the job has been returned, we recommend that MAILPROGRAM be configured. Moab will send an email to the user if a job is rejected.

Because the job is not fully processed, some attributes might change after the job has been submitted. For example, when a job class is remapped, the new class is not reflected until Moab begins its next scheduling iteration. Additionally, job arrays are not instantiated until Moab begins its next scheduling cycle.

If ENABLEHIGHTHROUGHPUT is TRUE, you must set NODEALLOCATIONPOLICY to FIRSTAVAILABLE.

I.2.4 Load All Non-Completed Jobs at Startup

Use the LOADALLJOBCP parameter. By default, Moab loads non-complete jobs for active resource managers only. By setting LOADALLJOBCP to TRUE, Moab will load all non-complete jobs from all checkpoint files at startup, regardless of whether their corresponding resource manager is active.

I.2.5 Reducing Job Start Time

Use the ASYNCSTART parameter. By default, Moab will launch one job at a time and verify that each job successfully started before launching a subsequent job. For organizations with large numbers of very short jobs (less than 2 minutes in duration), the delay associated with confirming successful job start can lead to productivity losses. If tens or hundreds of jobs must be started per minute, and especially if the workload is composed primarily of serial jobs, then the resource manager ASYNCSTART flag can be set. When set, Moab will launch jobs optimistically and confirm success or failure of the job start on the subsequent scheduling iteration. Also consider adding the ASYNCDELETE flag if users frequently cancel jobs.

I.2.6 Reducing Job Reservation Creation Time

Use the RMCFGJOBRSVRECREATE attribute. By default, Moab destroys and re-creates job reservations each time a resource manager updates any aspect of a job. Historically, this stems from the fact that certain resource managers would inadvertently or intentionally migrate job tasks from originally requested nodes to other nodes. To maintain synchronization, Moab re-creates reservations each iteration therefore incorporating these changes. On most modern resource managers, these changes never occur, but the effort required to handle this case grows with the size of the cluster and the size of the queue. Consequently, on very large systems with thousands of nodes and thousands of jobs, a noticeable delay is present. By setting `JOBRSVRECREATE` to `FALSE` on resource managers that do not exhibit this behavior, significant time savings per iteration can be obtained.

I.2.7 Optimizing Backfill Time

Use the OPTIMIZEDBACKFILL flag. Speeds up backfill when a system reservation is in use.

I.2.8 Constraining Moab Logging - LOGLEVEL

Use the LOGLEVEL parameter. When running on large systems, setting `LOGLEVEL` to 0 or 1 is normal and recommended. Only increase `LOGLEVEL` above 0 or 1 if you have been instructed to do so by Moab support.

I.2.9 Preemption

When preemption is enabled Moab can take considerably more time scheduling jobs for every scheduling iteration. Preemption increases the number of options available to Moab and therefore takes more time for Moab to optimally place jobs. If you are running a large cluster or have more than the usual amount of jobs (>10000), consider disabling preemption. If disabling preemption is not possible, consider limiting its scope to only a small subset of jobs (as both preemptors and preemptees).

I.2.10 Handling Transient Resource Manager Failures

Use the `RMCFG` MAXITERATIONFAILURECOUNT attribute.

I.2.11 Constrain the Number of Jobs Preempted Per Iteration

Use the JOBMAXPREEMPTPERITERATION parameter.

For very large job count systems, configuration options controlling the maximum supported limits might need to be adjusted, including the maximum number of reservations and the maximum number of supported evaluation ranges.

I.2.12 Scheduler Settings

If using Moab, there are a number of parameters that can be set on the scheduler, which may improve Torque performance. In an environment containing a large number of short-running jobs, the `JOBAGGREGATIONTIME` parameter can be set to reduce the number of workload and resource queries performed by the scheduler when an event based interface is enabled. Setting `JOBAGGREGATIONTIME` instructs the scheduler to ignore events coming from the resource manager and perform scheduling at regular intervals, rather than around resource manager events. If the *`pbs_server`* daemon is heavily loaded and PBS API timeout errors (i.e., 'Premature end of message') are reported within the scheduler, the `TIMEOUT` attribute of the `RMCFG` parameter can be set with a value of between 30 and 90 seconds.

I.2.13 Configure Torque for Large Job Numbers

Torque's `use_job_subdirs` server parameter enables Torque to handle large numbers of jobs more efficiently. For more information, see 'use_jobs_subdirs' in the *Torque Resource Manager Administrator Guide*.

I.3 Handling Large Numbers of Nodes

For very large clusters (>= 10,000 processors) default scheduling behavior might not scale as desired. To address this, the following parameters should be considered:

Parameter	Recommended Setting
RMPOLLINTERVAL	In large node environments with large and long jobs, scheduling overhead can be minimized by increasing `RMPOLLINTERVAL` above its default setting. If an event-driven resource management interface is available, values of two minutes or higher can be used. Scheduling overhead can be determined by looking at the scheduling load reported by mdiag -S.
LIMITEDNODECP	Startup/shutdown time can be minimized by disabling full node state checkpointing that includes some statistics covering node availability.
SCHEDCFG FLAGS=" FASTRSVSTARTUP	When you have reservations on a large number of nodes, it can take Moab a long time to recreate them on startup. Setting the `FASTRSVSTARTUP` scheduler flag greatly reduces startup time.

* For clusters where the number of nodes or processors exceeds 50,000, the maximum stack size for the shell in which Moab is started might need to be increased (as Moab might

crash if the stack size is too small). On most UNIX/Linux based systems, the command *`ulimit -s unlimited`* can be used to increase the stack size limit before starting Moab. This can be placed in your Moab startup script.

See Appendix D for more information on default and supported object limits.

Avoid adding large numbers of NODECFG lines in the `moab.cfg` or `moab.d/*.cfg` files to keep the Moab boot time low.

For example, adding a configuration line to define features for each node in a large cluster (such as `NODECFG[x] Features+=green,purple`) can greatly increase the Moab boot time. If Moab processes 15 node configuration lines per second for a 50,000-node system, it could add approximately 55 minutes of node configuration processing to the Moab boot time.

In this case, it is better to define the node features in the resource manager configuration.

I.4 Handling Large SMP Systems

For large-way SMP systems (> 512 processors/node) Moab defaults might need adjustment.

Parameter	Recommended Setting
MAXRSVPERNODE	By default, Moab does not expect more than 64 jobs per node to be running or have future reservations. Increasing this parameter to a value larger than the expected maximum number of jobs per node is advised.

I.5 Resource Manager Scaling

Proper Resource Manager Configuration

- Torque
  - See 'Large Cluster Considerations' in the *Torque Resource Manager Administrator Guide*.

I.6 Server Sizing

See 'Hardware Requirements' in the *Moab HPC Suite Installation and Configuration Guide* for recommendations.

Appendix J: Configuring Moab as a Service

Scripts that follow can be used to start up Moab services automatically upon a reboot. To enable a service script, copy the script to `/etc/rc.d/init.d/S97moab`, edit the file to make needed localization changes (adjust binary paths, execution user, etc.), and add links to the `rc3.d`and `rc5.d` directories as in the example below:

```
> cp mwm.service /etc/rc.d/init.d/S97moab
> vi /etc/rc.d/init.d/S97moab
    (make needed localizations)
> ln -s /etc/rc.d/init.d/S97moab /etc/rc.d/rc3.d
> ln -s /etc/rc.d/init.d/S97moab /etc/rc.d/rc5.d
```

In this appendix:

J.1 Moab Grid Scheduler Service Script

Sample Script

```
#!/bin/sh
#
# moab          This shell script takes care of starting and stopping
#               the Moab Workload Manager.
#

# Source function library.
if [ -f /etc/init.d/functions ] ; then
  . /etc/init.d/functions
elif [ -f /etc/rc.d/init.d/functions ] ; then
  . /etc/rc.d/init.d/functions
else
  exit 0
fi

# Read in the command arguments
case "$1" in
  start)
       # Next start Moab Workload Manager...
       echo -n $"Starting Moab Workload Manager: "
       daemon su -l -s /bin/sh root -c "/opt/moab/sbin/moab"
       echo
       ;;

  stop)
       # Stop Moab
       echo -n $"Shutting down Moab Workload Manager: "
       killproc moab
       echo
       ;;

  restart)
       $0 stop
       $0 start
       ;;
  *)
       echo $"Usage: moab {start|stop|restart}"
       exit 1
esac

exit 0
```

J.2 Moab Workload Manager Service Scripts

In this section:

J.2.1 Moab Workload Manager Script

```
#!/bin/sh
#
# moab         This shell script takes care of starting and stopping
#              the Moab Workload Manager.
#

# Source function library.
if [ -f /etc/init.d/functions ] ; then
  . /etc/init.d/functions
elif [ -f /etc/rc.d/init.d/functions ] ; then
  . /etc/rc.d/init.d/functions
else
  exit 0
fi

# Read in the command arguments
case "$1" in
  start)
       # Next start Moab Workload Manager...
       echo -n $"Starting Moab Workload Manager: "
       daemon su -l -s /bin/sh root -c "/opt/moab/sbin/moab"
       echo
       ;;

  stop)
       # Stop Moab
       echo -n $"Shutting down Moab Workload Manager: "
       killproc moab
       echo
       ;;
  restart)
       $0 stop
       $0 start
       ;;
  *)
       echo $"Usage: moab {start|stop|restart}"
       exit 1
esac

exit 0
```

J.2.2 Moab Workload Manager + Torque Script

```
#!/bin/sh
#
# moab          This shell script takes care of starting and stopping
#               the TORQUE resource manager and Moab Workload Manager.
#
# description: TORQUE is a scalable resource manager which manages jobs in
#                 cluster environments. Moab is a cluster scheduler which uses
#                   TORQUE to schedule jobs on that cluster.
#

# Source function library.
if [ -f /etc/init.d/functions ] ; then
  . /etc/init.d/functions
elif [ -f /etc/rc.d/init.d/functions ] ; then
  . /etc/rc.d/init.d/functions
else
  exit 0
fi

# Read in the command arguments
case "$1" in
  start)
       # Start TORQUE services first...

       echo -n $"Starting TORQUE services: "
       daemon /usr/local/torque/sbin/pbs_mom
       daemon /usr/local/torque/sbin/pbs_server
       echo

       # Next start Moab Workload Manager...
       echo -n $"Starting Moab Workload Manager: "
       daemon su -l -s /bin/sh root -c "/opt/moab/sbin/moab"
       echo
       ;;

  stop)
       # Stop Moab first...

       echo -n $"Shutting down Moab Workload Manager: "
       killproc moab
       echo

       echo -n $"Shutting down TORQUE services: "
       killproc pbs_server
       echo
       killproc pbs_mom
       echo
       ;;

  restart)
       $0 stop
       $0 start
       ;;
  *)
       echo $"Usage: moab {start|stop|restart}"
       exit 1
esac

exit 0
```

Appendix K: Migrating from Maui 3.2

This guide is intended to help facilitate migrating from Maui to Moab. If you do not have Moab yet, you can download a free evaluation version. At a high level, migrating from Maui 3.2 to Moab involves minimal effort. In fact, Moab fully supports all Maui parameters and commands. Migration can consist of nothing more than renaming `maui.cfg` to `moab.cfg` and launching Moab using the *`moab`* command. With this migration, the biggest single issue is becoming aware of all the new facilities and capabilities available within Moab. Beyond this, migration consists of a few minor issues that might require attention such as some statistics and priorities.

Another approach of migrating from Maui to Moab is to configure Moab in Monitor mode and run it beside Maui. Maui will continue to perform the scheduling and control workload. Moab will simply monitor the cluster environment using the policies configured in `moab.cfg`. Moab will not have the ability to affect workload, providing a safe and risk-free environment to evaluate Moab without affecting your production environment. You can also have Moab capture resource and workload trace files and allow Moab to simulate what it would have done if it controlled workload. When you feel comfortable with and want to run Moab live on your cluster, all you need to do is change the mode to NORMAL, stop Maui, and restart Moab. Current jobs will remain running and Moab will take over control of scheduling.

As with any migration, we suggest that you back up important files such as the following: `maui.cfg`, `maui.log` and `maui.ck`.

In this appendix:

K.1 Migrating from Maui to Moab

1. Install Moab.
2. Copy your `maui.cfg` file to the `MOABHOMEDIR/etc` (`/opt/moab/etc`) and rename it `moab.cfg`.
3. Stop Maui.
4. Start Moab.

5. If applicable, re-apply those configurations found in the Statistics and Checkpointing section that need adjustment after migration and any parameters in `moab.cfg` that point to a Maui file like `maui.log`.

K.2 Other Notes

This section provides more information on the minor differences between Maui and Moab, and the changes you might need to make.

In this section:

K.2.1 File Naming

Moab uses slightly different naming than Maui. The following table displays these changes:

File	Maui	Moab
executable	maui	moab
logs	`maui.log`	`moab.log`
configuration file	`maui.cfg`	`moab.cfg`

K.2.2 Statistics and Checkpointing

Moab supports Maui version 3.2 or higher workload statistics, allowing it to process historical statistics based on these statistics files. No changes are required to use these statistics.

Moab does not support the Maui 3.2 checkpointing format. Because of this, state information checkpointed under Maui will not be available at the time of the migration. The loss of this information will have the following impact:

- Admin reservations, if any, will need to be re-created.
- Processed credential and scheduler statistics (displayed by *`showstats`*) will be lost.

- Admin job system priority configured by the *setspri* command and QoS assignments configured by the *setqos* command, if any, will be lost.

K.2.3 Verify Configuration File Compatibility

The command mdiag -C will perform diagnostics on your new configuration file and might prove helpful in identifying any issues.

K.2.4 Environment Variables

Scheduler environment variables are supported under Moab with obvious naming changes. Sample environment variables follow:

Maui	Moab
MAUIHOMEDIR	MOABHOMEDIR
MAUIDEBUG	MOABDEBUG
MAUICRASHVARIBALE	MOABCRASHVARIABLE
MAUIENABLELOGBUFFERING	MOABENABLELOGBUFFERING
MAUIRECOVERYACTION	MOABRECOVERYACTION
MAUI-COMMANDS-PATH	MOAB-COMMANDS-PATH
MAUIENABLELOGBUFFERING	MOABENABLELOGBUFFERING

K.3 Running Maui and Moab Side-By-Side

1. Install Moab on your cluster. **Note:** Installation steps will differ slightly from a typical Moab installation.
   a. Run *./configure*.
   b. Run *make*.
   c. You will need to set your `MOABHOMEDIR` environment variable to the location where you built Moab by typing `export MOABHOMDIR=[make directory]`.

2. To have Moab use all the same policies as Maui, copy `maui.cfg` to the MOAB`HOMEDIR/etc` and rename it `moab.cfg`. **Note:** You can also start your `moab.cfg` file from scratch. Just use the `moab.cfg` already in the `MOABHOMEDIR/etc`.
3. Make sure that the port in `moab.cfg` is different than the port used in `maui.cfg`.
4. In the `moab.cfg` file, add the parameter, `SERVERMODE=MONITOR`. **Note:** If you used the `moab.cfg` from scratch, on the `SCHEDCFG` line add `MODE=MONITOR`.
5. You will need to either put the Moab commands in your environment path (located in `MOABHOMEDIR/bin`) or run the commands from their location if you still want to use the Maui commands in your environment path.
6. Run Moab using the *`moab`* command located in `MOABHOMEDIR/bin`.

Appendix L: Node Allocation Plug-in Developer Kit

Each time Moab schedules a job, it must choose the nodes on which the job will run. Moab uses the Node Allocation policy to select the available nodes to be used. Because there are so many different systems and cluster topologies, you now have the ability to create and use a node allocation plug-in for allocating nodes based on your cluster's interconnect topology.

The plug-in policy enables you to write your own algorithm to choose which nodes will be used. This algorithm is contained in a shared library that Moab loads at run time.

To obtain the Plug-in Developer Kit (PDK) with the header file and example code, contact your sales representative.

In this appendix:

L.1 Moab Configuration

The actual loading of a plug-in is accomplished by specifying the plug-in in the Moab configuration file, `moab.cfg`.

In this section:

L.1.1 Moab.cfg

We recommend that you store all Moab plug-ins in the `$MOABHOMEDIR/lib` directory (e.g., `/opt/moab/lib`) as shared libraries (`*.so`). The name of the actual plug-in shared library file is up to the plug-in developer, which means you must give the correct name in the moab.cfg file to form the absolute plug-in filename.

If a plug-in's specified shared library filename starts with a forward slash (/), it is an absolute file path name and Moab simply uses it without alteration. For example, if a plug-in's specified shared library filename is `/opt/moab/plugins/plugin.so`, Moab will use it as the absolute plug-in file path name.

If a plug-in's specified shared library filename does not start with a forward slash (/), it is a plug-in name and Moab forms the plug-in's absolute path name by concatenating the Moab home directory, '/lib/lib', the specified plug-in name, and '.so' to obtain the absolute path name. For example, if the `$MOABHOMEDIR` environment variable contains `/opt/moab` and the plug-in name is `plugin`, Moab will create `/opt/moab/lib/libplugin.so` and use it as the absolute plug-in file path name.

L.1.2 Syntax Rules

In order for Moab to use a plug-in for the Node Allocation policy, instead of a built-in Moab policy, you must configure the policy in the moab.cfg file with the value 'PLUGIN:' followed by the plug-in's shared library file name. The examples below assume the environment variable `$MOABHOMEDIR` has a value of `/opt/moab`. Note the use of relative and absolute plug-in shared library file path names in the parameter value and how they affect Moab's construction of the full path name.

Partition	Plug-in Name	moab.cfg Parameter	Moab-derived Full Path Name
global	`plugin.so`	`NODEALLOCATIONPOLICY PLUGIN:plugin.so`	`/opt/moab/lib/libplugin.so`
global	`/usr/local/plugins/plugin.so`	`NODEALLOCATIONPOLICY PLUGIN:/usr/local/plugins/plugin.so`	`/usr/local/plugins/plugin.so`
abc	`plugin.so`	`PARCFG[abc] NODEALLOCATIONPOLICY =PLUGIN:plugin.so`	`/opt/moab/lib/libplugin.so`
xyz	`/usr/local/plugins/plugin.so`	`PARCFG[xyz] NODEALLOCATIONPOLICY= PLUGIN:/usr/local/plugins/plugin.so`	`/usr/local/plugins/plugin.so`

L.1.3 Troubleshooting

The following commands can be used to confirm that the plug-in Node Allocation Policy was loaded properly.

mschedctl -l

`mschedctl -l` is used to print out Moab's in memory configurations. If the plug-in policy, with its full path, doesn't show for the configured partition then Moab failed to load the partition. Note that when the `NODEALLOCATIONPOLICY` is configured globally, it is configured on the 'ALL' partition.

```
$ mschedctl -l -v|grep ^NODEALLOCATIONPOLICY
NODEALLOCATIONPOLICY[ALL] PLUGIN:/opt/moab/lib/libfirstavailable.so
NODEALLOCATIONPOLICY[a] PLUGIN:/opt/moab/lib/liblastavailable.so
NODEALLOCATIONPOLICY[b] CONTIGUOUS
NODEALLOCATIONPOLICY[c] PLUGIN:/opt/moab/lib/libfirstavailable.so
NODEALLOCATIONPOLICY[d] [NONE]
```

mdiag -C

`mdiag -C` is used to validate the moab.cfg configuration. With a plug-in node allocation policy, Moab will validate that it can successfully load the plug-in and that all of the required symbols are present.

```
$ mdiag -C
...
INFO: line #35 is valid: 'NODEALLOCATIONPOLICY PLUGIN:firstavailable'
INFO: line #36 is valid: 'PARCFG[a]NODEALLOCATIONPOLICY=PLUGIN:lastavailable'
INFO: line #37 is valid: 'PARCFG[b]NODEALLOCATIONPOLICY=CONTIGUOUS'
INFO: line #38 is valid: 'PARCFG[d]NODEALLOCATIONPOLICY=PLUGIN:firstavailable'
```

L.2 Writing the Plug-In

In this section:

L.2.1 Node Allocation Plug-in

A plug-in is a shared library that has specific functions and variables that will be called directly from Moab. The plug-in conforms to a C language API. The API is specified through an include file: `moab-plugin.h`. This file must be included in the plug-in code. The include file provides function definitions, structures and variables that will be used when communicating with Moab.

When you write the plug-in, you need to ensure that the plug-in code is robust. If the plug-in crashes, Moab will crash. You will need to handle your own memory appropriately. If the plug-in has memory leaks, Moab will have similar issues. If you want to maintain logs, the plug-in will need to be responsible for its own logging.

L.2.2 API and Data Structures

The Application Programmer Interface (API) for the Moab Node Allocation plug-in consists of three data items and three entry points that must be supplied to Moab by the plug-in:

Plug-in Supplied Data	Description
const char *PLUGIN_NAME = "Node Allocation plugin 1.1";	This character pointer is used by Moab when logging information regarding the operation of the plug-in.
const char *PLUGIN_TYPE = PLUGIN_TYPE_ NAME_ NODEALLOCATION;	This character pointer is used by Moab to verify the type of plug-in. The value of this data is supplied by the `moab-plugin.h` source file. The plug-in must set this as shown so that Moab does not attempt to use a plug-in incorrectly. Moab uses this to determine whether the plug-in API type is correct and to allow Moab to correctly communicate with the plug-in.
const char *PLUGIN_VERSION = PLUGIN_API_ VERSION;	This character pointer is used by Moab to verify the API version number. The value of this data is supplied by the `moab-plugin.h` source file. The plug-in must set this as shown so that the correct version of the `moab-plugin.h` is supplied to Moab. Moab uses this to determine whether the API version is correct and to allow Moab to correctly communicate with the plug-in.

Load Time API	Description
initialize()	`int initialize(const char *name, void **data_handle)` The plug-in must supply an initialize() entry point. This entry point is called for each use instance of the plug-in. For example, if the plug-in is used on two different partitions, the initialize() entry point will be called once for each partition. • Name — The name is the unique identifier that is used to distinguish multiple instances of the plug-in and for logging. When configured globally, the name "ALL" will be given. • Data handle — The data_handle points to a location where the plug-in should store a pointer to any internal data needed by the plug-in between calls to the API. The actual format and structure of the data is up to the plug-in. Moab will supply this pointer back to the plug-in each time a plug-in entry point is called. This data can provide context for the plug-in usage instance.
Return codes	The initialize() entry point should return one of two return statuses as defined in `moab-plugin.h`:

Load Time API	Description

	`#define PLUGIN_RC_SUCCESS 0`
	`#define PLUGIN_RC_FAILURE 1`

Gathering node info	The initialize() entry point must gather any information about system nodes, their topology, interconnection, and configuration that it needs to make correct node allocations. Since Moab does not know what information the plug-in might need, the plug-in must gather this information itself.
Memory considerations	The plug-in may allocate memory for temporary or persistent data as needed, but *must* de-allocate or return the memory when finished. Not returning memory can result in memory leaks and unstable operation on the part of Moab.
Multiple access	A given loaded plug-in can be used by more than one partition. This means that the plug-in must maintain its internal data in such a way that calls to the plug-in for the separate partitions do not conflict. We recommend that internal data be allocated and a pointer to the data be kept in the data_ handle described above as opposed to using global or static variables. Any global or static data will be shared between possible multiple instances of the plug-in.

Runtime API	Description

node_ allocate ()	`int node_allocate (`
	`  void                *data_handle,`
	`  const char          *job_name,`
	`  int                 container_count,`
	`  nalloc_container_t  container[])`
	
	The plug-in must provide a node_allocate() entry point. This entry point is called each time Moab needs to determine where (on what nodes) a job will eventually run. Note that this entry point can be called many times before the job is actually scheduled to run.
	
	• Data structures — Moab uses C data structures to pass information and lists of nodes to the plug-in and receive them back from the plug-in. See `moab-plugin.h` for the definitions of these structures and for information on how they relate to one another.

Operations	A node allocation request consists of one or more requirements. Each of these requirements is provided within a 'container' structure. The container has information regarding the requirement to be met, the count and list of all nodes that are available to meet the requirement and a place to return the list of nodes that the plug-in has chosen to use for the job.

Runtime API	Description

Command	Moab Job Task Count	Job Node Count	Job Tasks Per Node	Node CFG Procs	Node AVL Procs	Plug-in Node Mapped TC	requirement ->taskcount	return_node_count
Non-ExactNode								
-l nodes=12	12	0	0	8	8	8	12	2
-l nodes=12:ppn=2	24	0	2	8	8	8	24	3
ExactNode								
-l nodes=4	4	4	0	8	8	1	4	4
-l nodes=4:ppn=2	8	4	2	8	8	2	8	4
-l nodes=12	12	0	0	8	6	6	12	2

The duty of the plug-in is to use the information that it has previously gathered (during the initialization) to select from the available nodes those that will best fulfill the requirements.

The basic algorithm is to consume all the taskcount and memory on each node until the consumed task count is greater than or equal to the container's task_count and memory requirements.

A job's taskcount is calculated differently based on the JOBNODEMATCHPOLICY parameter. By default, it isn't defined and `-l nodes=#` actually requests the number of tasks without respect to the number of nodes. In this case, the plug-in should consume all the tasks of each chosen node until the taskcount is greater and/or equal to the container's taskcount requirement. The plug-in is for node allocation and not task placement.

When the JOBNODEMATCHPOLICY EXACTNODE is configured, then `-l nodes=#` means the job wants # of nodes with 1 task per node. In this case, the nodes

Runtime API	Description
	passed to the plug-in will have a taskcount that is mapped down to what the job can only use on that node. Each node's taskcount should be consumed on each node until the summed amount is equal to the container's requirement taskcount requirement.

Errors and return codes	The plug-in may internally log any errors encountered and must return a success or error status as defined in `moab-plugin.h`:
	`#define    PLUGIN_RC_SUCCESS   0`
	`#define    PLUGIN_RC_FAILURE   1`

Multiple access safe	The node_allocate() entry point must support multiple access as described above.

Unload Time API	Description
finish()	`void finish(void *data_handle)` The plug-in must supply a finish() entry point. This entry point is called when Moab is preparing to disable and/or unload an instance of the plug-in.
Memory/resource cleanup	The plug-in must de-allocate and free up any resources acquired either during the initialize() entry point or during any calls to the node_allocate() entry point. When the last entry point returns, there should be no allocated memory or other resources still in use by the plug-in instance.
Multiple access safe	The finish() entry point must support multiple access as described above.

Appendix M: Scalable Systems Software Specification

In this appendix:

M.1 Scalable Systems Software Job Object Specification

SSS Job Object Specification
Release Version 3.1.0
26 April 2011

Scott Jackson, PNNL
David Jackson, Ames Lab
Brett Bode, Ames Lab

Status of this Memo

This document describes the job object to be used by Scalable Systems Software compliant components. It is envisioned for this specification to be used in conjunction with the SSSRMAP protocol with the job object passed in the Data field of Requests and Responses. Queries can be issued to a job-cognizant component in the form of modified XPATH expressions to the Get field to extract specific information from the job object as described in the SSSRMAP protocol.

Abstract

This document describes the syntax and structure of the SSS job object. A job model is described that is flexible enough to support the specification of very simple jobs and also jobs with elaborate and complex specification requirements in a way that avoids complex

structures and syntax when it is not needed. The basic assumption is that a solitary job specification should be usable for all phases of the job lifecycle and can be used at submission, queuing, staging, reservations, quotations, execution, charging, accounting, etc. This job specification provides support for multi-step jobs and also jobs with disparate task descriptions. It accounts for operational requirements in a grid or meta-scheduled environment where the job is executed by multiple hosts in different administrative domains that support different resource management systems.

Table of Contents

1.0 Introduction

This specification proposes a standard XML representation for a job object for use by the various components in the SSS Resource Management System. This object will be used in multiple contexts and by multiple components. It is anticipated that this object will be passed via the Data Element of SSSRMAP Requests and Responses.

1.1 Goals

There are several goals motivating the design of this representation.

The representation needs to be inherently flexible. We recognize we will not be able to exhaustively include the ever-changing job properties and capabilities that constantly arise.

The representation should use the same job object at all stages of that job's lifecycle. This object will be used at job submission, queuing, scheduling, charging and accounting, therefore it might need to distinguish between requested and delivered properties.

The design must account for the properties and structure required to function in a meta or grid environment. It needs to include the capability to support local mapping of properties, global namespaces, etc.

The equivalent of multi-step jobs must be supported. Each step (job) can have multiple logical task descriptions.

Many potential users of the specification will not be prepared to implement the complex portions or fine-granularity that others need. There needs to be a way to allow the more complicated structure to be added as needed while leaving more straightforward cases simple.

There needs to be guidance for how to understand a given job object when higher order features are not supported by an implementation, and which parts are required, recommended and optional for implementers to implement.

It needs to support composite resources.

It should include the ability to specify preferences or fuzzy requirements.

1.2 Non-Goals

Namespace considerations and naming conventions for most property values are outside of the scope of this document.

1.3 Examples

Example M-1: Very Simple Example

This example shows a simple job object that captures the requirements of a simple job:

```
<Job>
  <Id>PBS.1234.0</Id>
  <State>Idle</State>
  <User>scottmo</User>
  <Executable>/bin/hostname</Executable>
  <Processors>16</Processors>
  <Duration>3600</Duration>
</Job>
```

Example M-2: Moderate Example

This example shows a moderately complex job object that uses features such as required versus delivered properties:

```
<Job>
  <Id>PBS.1234.0</Id>
  <Name>Heavy Water</Name>
  <Project>nwchemdev</Project>
  <User>peterk</User>
  <Application>NWChem</Application>
  <Executable>/usr/local/nwchem/bin/nwchem</Executable>
  <Arguments>-input basis.in</Arguments>
  <InitialWorkingDirectory>/home/peterk</InitialWorkingDirectory>
  <Machine>Colony</Machine>
  <QualityOfService>BottomFeeder</QualityOfService>
  <Queue>batch_normal</Queue>
  <State>Completed</State>
  <StartTime>1051557713</StartTime>
  <EndTime>1051558868</EndTime>
  <Charge>25410</Charge>
  <Requested>
    <Processors op=”GE”>12</Processors>
    <Memory op=”GE” units=”GB”>2</Memory>
    <Duration>3600</Duration>
  </Requested>
  <Delivered>
    <Processors>16</Processors>
    <Memory metric="Average" units="GB">1.89</Memory>
    <Duration>1155</Duration>
  </Delivered>
  <Environment>
    <Variable name="PATH">/usr/bin:/home/peterk</Variable>
  </Environment>
</Job>
```

Example M-3: Elaborate Example

This example uses a job group to encapsulate a multi-step job. It shows this protocol's ability to characterize complex job processing capabilities. A component that processes this message is free to retain only that part of the information that it requires. Superfluous information can be ignored by the component or filtered out (by XSLT, for example).

```
<JobGroup>
  <Id>workflow1</Id>
  <State>Active</State>
  <Name>ShuttleTakeoff</Name>
  <JobDefaults>
    <StagedTime>1051557859</StagedTime>
    <SubmitHost>asteroid.lbl.gov</SubmitHost>
    <SubmitTime>1051556734</SubmitTime>
    <Project>GrandChallenge18</Project>
    <GlobalUser>C=US,O=LBNL,CN=Keith Jackson</GlobalUser>
    <User>keith</User>
    <Environment>
      <Variable name="LD_LIBRARY_PATH">/usr/lib</Variable>
      <Variable name="PATH">/usr/bin:~/bin:</Variable>
    <Environment>
  </JobDefaults>
  <Job>
    <Id>fr15n05.1234.0</Id>
    <Name>Launch Vector Initialization</Name>
    <Executable>/usr/local/gridphys/bin/lvcalc</Executable>
    <Queue>batch</Queue>
    <State>Completed</State>
    <Machine>SMP2.emsl.pnl.gov</Machine>
    <StartTime>1051557713</StartTime>
    <EndTime>1051558868</EndTime>
    <Quote>https://www.pnl.gov/SMP2#654321</Quote>
    <Charge units="USD">12.75</Charge>
    <Requested>
      <Duration>3600</Duration>
      <Processors>2</Processors>
      <Memory>1024</Memory>
    </Requested>
    <Delivered>
      <Duration>1155</Duration>
      <Processors consumptionRate="0.78">2</Processors>
      <Memory metric="Max">975</Memory>
    </Delivered>
    <TaskGroup>
      <TaskCount>2</TaskCount>
      <TaskDistribution type="TasksPerNode">1</TaskDistribution>
      <Task>
        <Node>node1</Node>
        <Process>99353</Process>
      </Task>
      <Task>
        <Node>node12</Node>
        <Process>80209</Process>
      </Task>
    </TaskGroup>
  </Job>
  <Job>
    <Id>fr15n05.1234.1</Id>
    <Name>3-Phase Ascension</Name>
    <Queue>batch_normal</Queue>
    <State>Idle</State>
    <Machine>Colony.emsl.pnl.gov</Machine>
    <Priority>1032847</Priority>
    <Hold>System</Hold>
    <StatusMessage>Insufficient funds to start job</StatusMessage>
    <Requested>
      <Duration>43200</Duration>
    </Requested>
    <TaskGroup>
      <TaskCount>1</TaskCount>
      <Name>Master</Name>
      <Executable>/usr/local/bin/stage-coordinator</Executable>
      <Memory>2048<Memory>
      <Resource name=”License” type=”ESSL2”>1</Resource>
      <Feature>Jumbo-Frame</Feature>
    </TaskGroup>
    <TaskGroup>
      <Name>Slave</Name>
      <TaskDistribution type=”Rule”>RoundRobin</TaskDistribution>
      <Executable>/usr/local/bin/stage-slave</Executable>
      <NodeCount>4</NodeCount>
      <Requested>
        <Processors group=”-1”>12</Processors>
        <Processors conj=”Or” group=”1”>16</Processors>
        <Memory>512</Memory>
        <Node aggregation=”Pattern”>fr15n.*</Node>
      </Requested>
    </TaskGroup>
  </Job>
</JobGroup>
```

2.0 Conventions Used in this Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 Table Column Interpretations

The columns of the property tables in this document have the following meanings:

Element Name	Name of the XML element (xsd;element) see [DATATYPES]
Type	Data type defined by xsd (XML Schema Definition) as: • String — xsd:string (a finite length sequence of printable characters). • Integer — xsd:integer (a signed finite length sequence of decimal digits). • Float — xsd:float (single-precision 32-bit floating point). • Boolean — xsd:boolean (consists of the literals 'true' or 'false'). • DateTime — xsd:int (a 32-bit unsigned long in GMT seconds since the EPOCH). • Duration — xsd:int (a 32-bit unsigned long measured in seconds).
Description	Brief description of the meaning of the property.
Appearance	An indication of whether the given property must appear in the parent element. It assumes the following meanings: • MUST — This property is REQUIRED when the parent is specified. • SHOULD — This property is RECOMMENDED when the parent is specified. • MAY — This property is OPTIONAL when the parent is specified.
Compliance	An indication of the relative importance of supporting the given property: • MUST — A compliant implementation MUST support this property. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.
Categories	Some properties can be categorized into one of several categories. Letters in this column indicate that the given property can be classified in the following property categories. • R — This property can be encompassed in a Requested element. • D — This property can be encompassed in a Delivered element.

2.3 Element Syntax Cardinality

Selected elements in the element syntax sections use regular expression wildcards with the following meanings:

Wildcard	Description
*	Zero or more occurrences.
+	One or more occurrences.

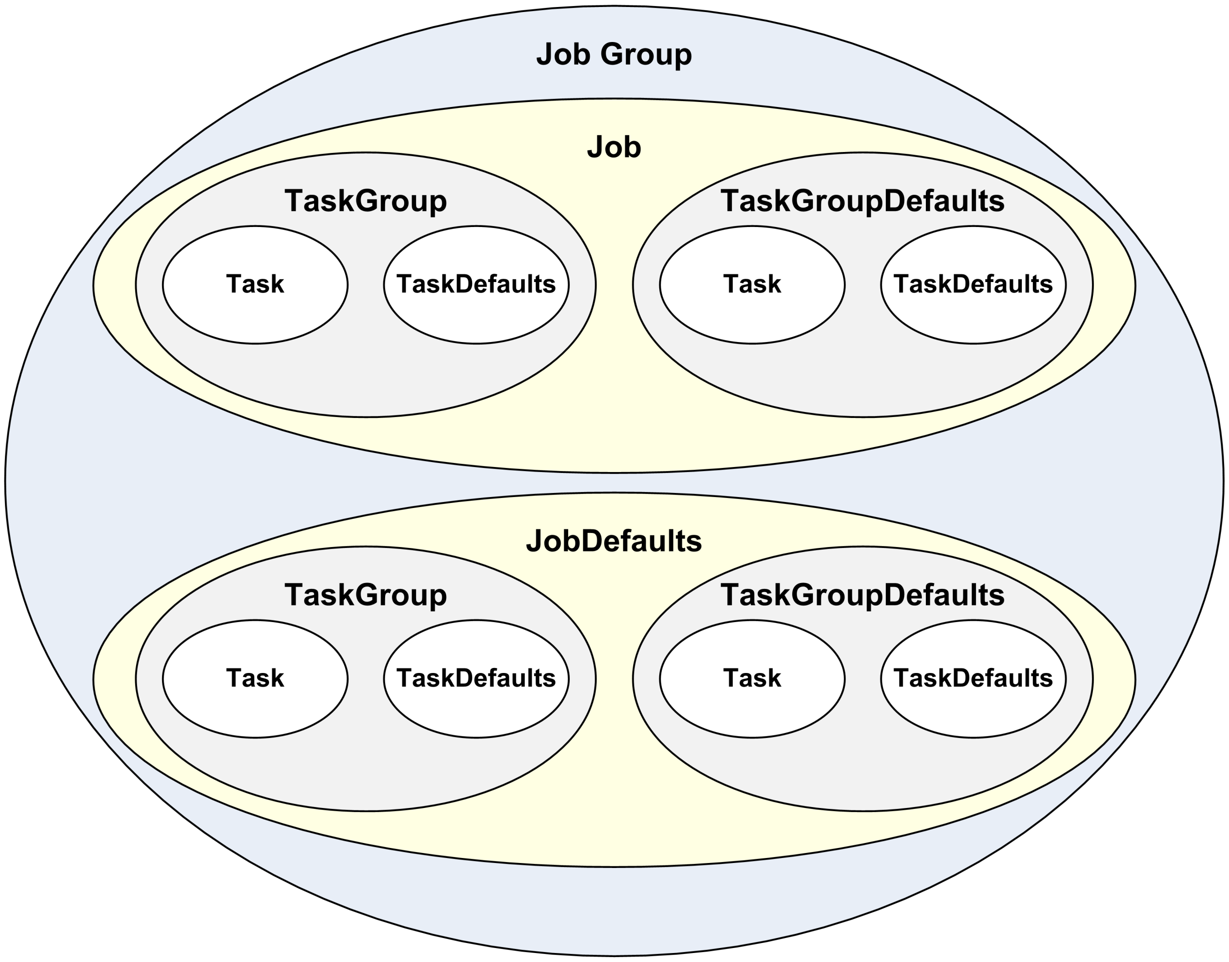
Wildcard	Description
?	Zero or one occurrences.

The absence of one of these symbols implies exactly one occurrence.

3.0 The Job Model

The primary object within the job model is a job. A job can be thought of as a single schedulable entity and will be the object normally seen in job queues.

Image M-1: JobGroup contains Job and JobDefaults, which contain TaskGroup and TaskGroupDefaults



Jobs with dependencies on other jobs can be submitted in a job group. Jobs within a job group form a DAG (directed acyclic graph) where the nodes are jobs and the edges represent dependencies on the status of previous jobs. A job group will consist of at least one job. A job group can optionally specify job defaults, which are a set of job properties to be assumed by all jobs within the job group unless overridden within the job.

A job can consist of multiple tasks, which are the finest grained work unit and represent an endpoint for executing a given process instance. For example, a job that requests 3 nodes and 4 processors will have 4 tasks, two on one node and one on each of two nodes. Tasks can be grouped into task groups, which are logical aggregations of tasks and their common properties. Submit filters, prologs, epilogs, notification scripts, etc. run once only for each job. Whereas task groups function as logical descriptions of tasks and their properties, they also describe the number of such tasks and the nodes that they run on. As an example, a master task group (consisting of a single task) might ask for a node with a MATLAB license, 2GB of memory and an Internet connected network adapter while a slave task group (consisting of 12 tasks) could be targeted for nodes with more CPU bandwidth -- all within the same job and utilizing a common MPI ring. Tasks (and therefore task groups) can have different executables or environments, specify different consumable resources or node properties. A job, therefore, can specify one or more task group. A job that does not specify an explicit task group is considered as having a single implicit task group. A job can optionally specify task group defaults, which are a set of task group properties to be assumed by all task groups within the job unless overridden within a task group.

A task group can specify one or more tasks. A task group that does not specify an explicit task is considered as having a single implicit task. A task group can optionally specify task defaults, which are a set of task properties to be assumed by all tasks within the task group unless overridden within a task.

4.0 JobGroup Element

A JobGroup is an optional element that aggregates one or more interdependent jobs. Some resource managers support the submission of job groups (multi-step jobs) and queries on the status of an entire job group.

- A compliant implementation MAY support this element.
- A JobGroup MUST specify one or more JobGroup Properties.
- A JobGroup MUST contain one or more Jobs.
- A JobGroup MAY contain zero or more JobsDefaults.

The following illustrates this element’s syntax:

```
<JobGroup>
  <!-- JobGroup Properties -->+
  <Job/>+
  <JobDefaults/>?
</JobGroup>
```

4.1 JobGroup Properties

JobGroup Properties are properties that apply to the job group as a whole. These include the job group ID, jobs and job defaults, and other simple optional job properties.

Simple JobGroup Properties

Simple (unstructured) job group properties are enumerated in the table below:

Table M-1: Simple JobGroup Properties

Element Name	Type	Description	Appearance	Compliance
CreationTime	DateTime	Date and time that the job group was instantiated.	MAY	MAY
Description	String	Description of the job group.	MAY	MAY
Id	String	Job group identifier.	MUST	MUST
Name	String	Name of the job group.	MAY	SHOULD
State	String	State of the job group as a whole. Valid states can include NotQueued, Unstarted, Active, and Completed.	MAY	SHOULD

Job

A job group MUST contain one or more jobs.

See the next section for element details.

JobDefaults

A job group MAY contain zero or one job defaults.

See the next section for element details.

4.2 JobGroup Reference

When a simple reference to a predefined job group is needed in an encapsulating element, a JobGroup element is used with the text content being the job group ID:

```
<JobGroup> workflow1</JobGroup>
```

5.0 Job and JobDefaults Element

The Job and JobDefaults elements are of the same structure. A Job element encapsulates a job and can be expressed as a standalone object. A JobDefaults element can only appear within a JobGroup and represents the defaults to be taken by all jobs within the job group. Job properties in Job elements override any properties found in a sibling JobDefaults element.

- A compliant implementation MUST support the Job element.
- A compliant implementation MAY support the JobDefaults element only if it supports the JobGroup element.
- A job MUST specify one or more Job Properties.
- One or more TaskGroup elements MAY appear at this level.
- Zero or one TaskGroupDefaults elements MAY appear at this level.

The following illustrates this element's syntax:

```
<Job>
  <!-- Job Properties -->+
  <TaskGroup/>*
  <TaskGroupDefaults/>?
</Job>
```

5.1 Job Properties

Job Properties apply to a particular job or as default properties to all jobs. They include the job ID, job credentials, task groups, task group defaults, and other simple optional properties.

Simple Job Properties

Simple (unstructured) job properties are enumerated in the table below:

Table M-2: Simple Job Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Application	String	Type of application such as Gaussian or Nwchem.	MAY	MAY	
Architecture	String	Type	MAY	MAY	RD

Element Name	Type	Description	Appearance	Compliance	Categories
		architecture for the nodes on which this job must run.			
Arguments	String	The arguments for the executable.	MAY	SHOULD	
Charge	Float	The amount charged for the job.	MAY	SHOULD	
Checkpointable	Boolean	Can this job be checkpointed?	MAY	MAY	
CpuDuration	Duration	Number of cpu seconds used by the job.	MAY	SHOULD	
DeadlineTime	DateTime	Date and time that a job must end by.	MAY	MAY	
EligibleTime	DateTime	Date and time that a job must start after.	MAY	MAY	
EndTime	DateTime	Date and time that a job ended (independent of success or failure).	MAY	MUST	
Executable	String	Executable. This can be an absolute	MAY	MUST	

Element Name	Type	Description	Appearance	Compliance	Categories
		or relative path or a URI.*			
ExitCode	Integer	Exit code for the job.	MAY	SHOULD	
GlobalJob	String	Globally unique job identifier (possibly in the form of a URI).	MAY	SHOULD	
Hold	String	Hold(s) on the job. There may be multiple instances of this element if there is more than one ld on the job.	MAY	SHOULD	
InitialWorking-Directory	String	Initial working directory.	MAY	SHOULD	
Interactive	Boolean	Is this an interactive job?	MAY	SHOULD	
Id	String	A local job identifier assigned to the job by the local resource manager.	MUST	MUST	
Name	String	Name of the job.	MAY	SHOULD	

Element Name	Type	Description	Appearance	Compliance	Categories
State	String	State of the job. Valid states can include Idle, Hold, Running, Suspended, or Completed.	MAY	MUST	
Type	String	Type of job. Meaning of this extension property is context specific.	MAY	MAY	
Machine	String	Name of the system or cluster that runs the job.	MAY	MUST	RD
Network	String	Type of network adapter required by the job.	MAY	MAY	RD
NodeCount	Integer	Number of nodes used by the job.	MAY	MUST	RD
OperatingSystem	String	Operating System required by the job.	MAY	MAY	RD
Partition	String	Name of the partition where the job should run.	MAY	MAY	RD

Element Name	Type	Description	Appearance	Compliance	Categories
Priority	Integer	Current queue priority (or rank)for the job.	MAY	SHOULD	
QualityOfService	String	Name of the Quality of Service (QoS).	MAY	SHOULD	RD
Queue	String	Name of the Queue (or class)that the job runs in.	MAY	SHOULD	RD
Quote	String	Identifier for a guaranteed charge rate quote obtained by the job.	MAY	MAY	
Reservation	String	Identifier for a reservation used by the job.	MAY	MAY	RD
ReservationTime	DateTime	Date and time that a reservation was placed for the job.	MAY	MAY	
ResourceManager Type	String	Type of resource manager required to run this job.	MAY	MAY	RD
Restartable	Boolean	Can this job be restarted?	MAY	MAY	
Shell	String	Specified the	MAY	MAY	

Element Name	Type	Description	Appearance	Compliance	Categories
		shell necessary to interpret the job script.			
StagedTime	DateTime	Date and time that a job was staged to the local resource management system.	MAY	MAY	
StartCount	Integer	Number of times the scheduler tried to start the job.	MAY	MAY	
StartTime	DateTime	Date and time that the job started.	MAY	MUST	
StatusMessage	String	Natural language message that can be used to provide detail on why a job failed, isn't running, etc.	MAY	SHOULD	
SubmitTIme	DateTime	Date and time that a job was submitted.	MAY	SHOULD	
SubmitHost	String	FQDN of host where the job was submitted from.	MAY	SHOULD	

Element Name	Type	Description	Appearance	Compliance	Categories
Suspendable	Boolean	Can this job be suspended?	MAY	MAY	
SuspendDuration	Integer	Number of seconds the job was in the Suspended state.	MAY	MAY	
TimeCategory	String	This allows the specification of shifts like PrimeTime for charging purposes.	MAY	MAY	
Duration	Duration	Number of seconds in the Running state.	SHOULD	MUST	RD

* The Executable can be a script or a binary executable. If it is already on the target system, it can be referenced by an absolute or relative pathname (relative to InitialWorkingDirectory). If it is passed with the job in a File object (see SSSRMAP), it can be referenced by an absolute or relative URI. An absolute URI would specify a URL where the file can be downloaded (like with wget). A relative URI is specified by preceding an identifier by a pound sign, as in:

```
<Executable>#Script</Executable>
```

It will be found in a File object included along with the Job object with the Script as an identifier, as in:

```
<File id=”Script”>echo hello world</File>
```

Feature Element

The Feature element connotes an arbitrary named feature of a node:

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times within a given set of Job Properties.

- This element is of type String.
- This element MAY have an `aggregation` attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the `aggregation` attribute if the Feature element is supported. Possible values for this attribute include:
  - List — a comma-separated list of features.
  - Pattern — a regular expression (perl5) matching desired features.
- If an `aggregation` attribute is specified with the value of List, this element MAY also have a `delimiter` attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following is an example of a feature element:

```
<Feature aggregation=”List”>feature1,feature2</Feature>
```

OutputFile Element

The `OutputFile` element specifies the name of the file to which the output stream (stdout) from the job will be written:

- This element’s character content is the name of the file. If this element is omitted or it is empty, then an appropriate output file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute, which is a comma-separated list of output redirection attributes of type String. A compliant implementation SHOULD support this attribute if `OutputFile` is supported. Possible values for this attribute include:
  - Append — opens the output file for append
  - Close — closes and discards the output stream
  - Flush — output is written to output file as it is generated
  - Keep — leave the output file on the execution host
  - Merge — merges the output stream into the error stream

Note that when using the `redirectList` attributes, the cumulative effect of the `ErrorFile` and `OutputFile` directives may be order dependent.

The following is an example of an `OutputFile` element:

```
<OutputFile redirectList=”Append”>~/myjob.out</OutputFile>
```

ErrorFile Element

The `ErrorFile` element specifies the name of the file to which the error stream (stderr) from the job will be written:

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate error file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute, which is a comma-separated list of error redirection attributes of type String. A compliant implementation SHOULD support this attribute if `ErrorFile` is supported. Possible values for this attribute include:
    - Close — closes and discards the error stream
    - Append — opens the error file for append
    - Flush — output is written to output file as it is generated
    - Keep — leave the output file on the execution host
    - Merge — merges the error stream into the output stream

Note that when using the `redirectList` attributes, the cumulative effect of the `ErrorFile` and `OutputFile` directives may be order dependent.

The following is an example of an `ErrorFile` element:

```
<ErrorFile redirectList=”Merge”></ErrorFile>
```

InputFile Element

The `InputFile` element specifies the name of the file from which the input stream (stdin) for the job will be read:

- This element's character content is the name of the file. If this element is omitted or it is empty, then an appropriate input file is auto-determined by the queuing system.
- This element MAY have a `redirectList` attribute, which is a comma-separated list of input attributes of type String. A compliant implementation SHOULD support this attribute if `InputFile` is supported. Possible values for this attribute include:
    - Close — closes and discards the input stream

The following is an example of an `InputFile` element:

```
<InputFile redirectList=”Close”></InputFile>
```

NotificationList Element

The `NotificationList` element specifies the job-related events or conditions for which a notification will be sent.

- This element's character content is a comma-separated list of events or conditions for which a notification should be sent. Possible values for the elements of this list include:
    - JobStart — send a notification when the job starts
    - JobEnd — send a notification when the job ends
    - All — send notifications for all notifiable events
    - None — do not send notifications for any events
- This element MAY have a `uri` attribute of type String, which indicates where the notification is to be sent. A compliant implementation MAY support this attribute if `NotificationList` is supported. The `uri` is in the format: `[scheme://]authority` with the scheme being smtp and the authority being an email address by default.

The following is an example of a `NotificationList` element:

```
<NotificationList uri=”smith@business.com”>JobStart,JobEnd</NotificationList>
```

ResourceLimitElement

The `ResourceLimit` element represents a resource limit with its name and value:

- This element MUST have a `name` attribute of type String. A compliant implementation MUST support the name attribute if ResourceLimit is supported.
- This element MAY have a `type` attribute of type String that can have the values `Hard` or `Soft`. If the limit is enforced by the operating system, a hard limit is one that cannot be increased once it is set while a soft limit can be increased up to the value of the hard limit. If the type attribute is omitted, both the soft and hard limits are set.
- This element's character content is the resource limit's value.

Some typical names include:

Name	Description
CoreFileSize	Maximum core file size.
CpuTime	CPU time in seconds.
DataSegSize	Maximum data size.
FileSize	Maximum file size.

Name	Description
MaxMemorySize	Maximum resident set size.
MaxProcesses	Maximum number of processes.
MaxSwap	Virtual memory limit.
MaxMemLock	Maximum locked-in-memory address space.
MaxProcessors	Maximum processors.
MaxMemory	Maximum memory.
MaxDisk	Maximum disk space.
MaxNetwork	Maximum network bandwidth.
MaxFileIO	Maximum file i/o.
OpenFiles	Maximum number of open files.
Stacksize	Maximum stack size.

The following is an example of a `ResourceLimit` element:

```
<ResourceLimit name="CPUTime">1000000</ResourceLimit>
```

Credentials

Credentials are a special group of job properties that characterize an authenticated token or ID. They can be categorized in both requested and delivered forms.

Credential job properties are enumerated in the table below:

Table M-3: Credential Job Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Project	String	Name of the Project or Charge Account.	MAY	SHOULD	RD
GlobalUser	String	Globally unique user identifier. This could	MAY	SHOULD	RD

Element Name	Type	Description	Appearance	Compliance	Categories
		be an X.509 DN, for example.			
Group	String	Name of the local group ID.	MAY	MAY	RD
User	String	Name of the local user ID for the job.	MAY	MUST	RD

Environment Element

The Environment element encapsulates environment variables:

- This element MAY have an export attribute of type Boolean, which if set to `True`, indicates that all environment variables in the context of the job submission process should be exported in the job's execution environment.
- A compliant implementation SHOULD support this element.
- An Environment element MAY appear zero or one times within a given set of Job (or TaskGroup) Properties.
- An Environment element MAY contain one or more Variable elements.

The following illustrates this element's syntax:

```
<Environment>
  <Variable/>+
</Environment>
```

Variable Element

The Variable element represents an environment variable with its name and value.

This element MUST have a `name` attribute of type String. A compliant implementation MUST support the name attribute if Variable is supported. This element's character content is the environment variable's value.

The following is an example of a Variable element:

```
<Variable name="PATH">/usr/bin:/home/sssdemo</Variable>
```

Node Element

The Node element represents a node:

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times within a given set of Job Properties.
- This element is of type String.
- This element MAY have an `aggregation` attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the `aggregation` attribute if the Feature element is supported. Possible values for this attribute include:
  - List - a comma-separated list of features.
  - Pattern - a regular expression (perl5) matching desired features.
  - Range - a range of nodes of the form: <prefix>[5-23,77].
- If an `aggregation` attribute is specified with the value of List, this element MAY also have a `delimiter` attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.
- This element MAY have a `count` attribute of type Integer that indicates the instance count of the specified node(s).
- This element MAY be categorized as a requested or delivered property by being encompassed by the appropriate element.

The following is an example of a Node element:

```
<Node aggregation=”Pattern”>node[1-5]</Node>
```

TaskDistribution Element

The `TaskDistribution` element describes how tasks are to be mapped to nodes. This mapping can be expressed as a rule name, a task per node ratio or an arbitrary geometry.

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or one times in a given set of Job (or TaskGroup) Properties.
- This element is of type String.
- This element MAY have a `type` attribute of type String that provides a hint as to the type of mapping guidance provided. It can have values including `Rule`, `TasksPerNode`, `ProcessorsPerTask` or `Geometry`. A compliant implementation MAY support the `type` attribute if the `TaskDistribution` element is supported.
- It is possible to use `Processors`, `NodeCount` and `TaskCount` elements to specify a set of mutually contradictory task parameters. When this occurs, components are responsible for resolving conflicting requirements.

The following are three examples of a `TaskDistribution` element:

```
<TaskDistribution type="TasksPerNode">2</TaskDistribution>
<TaskDistribution type="Rule">RoundRobin</TaskDistribution>
<TaskDistribution type="Geometry">{1,4}{2}{3,5}</TaskDistribution>
```

Dependency Element

The Dependency element allows a job's execution to depend on the status of other jobs. In a job group (multi-step job), some jobs may delay execution until the failure or success of other jobs creating in general a Directed Acyclic Graph relationship between the jobs. This element's content is of type String and represents the job that the current job is dependent upon. Since a job can have two or more dependencies, this element can appear more than once in a given job scope. A compliant implementation SHOULD support this element if job groups are supported:

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times in a given set of Job (or TaskGroup) Properties.
- This element is of type String and contains the JobId that the current job is dependent upon.
- This element MAY have a `condition` attribute of type String that indicates the basis for determining when the current job executes in relation to the specified job. A compliant implementation MUST support this attribute if this element is supported. Possible values for this attribute include:
    - OnSuccess this job should run after the referenced job only if it completes successfully (this is the default if the `type` attribute is omitted).
    - OnFailure this job should run after the referenced job only if it fails.
    - OnExit this job should run after the referenced job exits.
- If the `condition` attribute is equal to OnExit, this element MAY have a `code` attribute of type Integer that indicates the exit code that will trigger this job to run. If the code attribute is omitted, then the current job should run after the referenced job for any exit status.
- This element MAY have a `designator` attribute of type String that indicates that indicates the property of the job that identifies it as the dependent job. A compliant implementation MAY support this attribute if this element is supported. Possible values for this attribute include:
    - JobId the job this job is dependent upon is specified by JobId (this is the default if the designator attribute is omitted).
    - JobName the job(s) this job is dependent upon are specified by JobName.

The following is an example of a Dependency element:

```
<Dependency condition="OnSuccess" designator="JobId">PBS.1234.0</Dependency>
```

Consumable Resources

Consumable Resources are a special group of properties that can have additional attributes and can be used in multiple contexts. In general a consumable resource is a resource that can be consumed in a measurable quantity.

- A consumable resource MAY have a `context` attribute of type String that indicates the sense in which the resource is used. A compliant implementation MAY support this attribute. Possible values for this attribute include:
  - Configured — run this task only on nodes having the specified configured resources.
  - Available — run this task only on nodes having the specified available resources. (this is the default if the `context` attribute is omitted).
  - Used — the task used the indicated resources (this is analogous to being including in a Delivered block).
  - Dedicated — the indicated amount of the resource should be dedicated to the task.
- A consumable resource MAY have a `units` attribute that is of type String that specifies the units by which it is being measured. If this attribute is omitted, a default unit is implied. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `metric` attribute that is of type String that specifies the type of measurement being described. For example, the measurement can be a Total, an Average, a Min or a Max. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `duration` attribute of type Duration that indicates the amount of time for which that resource was used. This need only be specified if the resource was used for a different amount of time than the duration for the job. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a `consumptionRate` attribute of type Float that indicates the average percentage that a resource was used over its duration. For example, an overbooked SMP running 100 jobs across 32 processors might want to scale the usage and charge by the average fraction of processor usage actually delivered. A compliant implementation MAY support this attribute if the element is supported.

- A consumable resource MAY have a `dynamic` attribute of type Boolean that indicates whether the resource allocated for this job should be allowed to grow or shrink dynamically. For example, if processors is specified with dynamic equal to True, the job can be dynamically allocated more processors as they become available. The growth bounds can be indicated via the `op` attribute, which is inherited when a consumable resource element is encapsulated within a *Requested* element. A compliant implementation MAY support this attribute if the element is supported.

Simple consumable resources are listed in the table below:

Table M-4: Simple Consumable Resources

Element Name	Type	Description	Appearance	Compliance	Categories
Disk	Float	Amount of disk.	MAY	SHOULD	RD
Memory	Float	Amount of memory.	MAY	SHOULD	RD
Network	Float	Amount of network.	MAY	MAY	RD
Processors	Integer	Number of processors.	MAY	MUST	RD
Swap	Float	Amount of virtual memory.	MAY	MAY	RD

The following are two examples for specifying a consumable resource:

```
<Memory metric=”Max” units=”GB”>483</Memory>
<Processors duration=”1234” consumptionRate=”0.63”>4</Processors>
```

Resource Element

In addition to the consumable resources enumerated in the above table, an extensible consumable resource is defined by the Resource element:

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times within a given set of job (or task group) properties.
- Like the other consumable resources, this property MAY be categorized as a requested or delivered property by being encompassed in the appropriate element.
- This element is of type Float.

- This element shares the *same properties and attributes as the other consumable resources* but it requires an additional `name` (and optional type) attribute to describe it.
- It MUST have a `name` attribute of type String that indicates the type of consumable resource being measured. A compliant implementation MUST support this attribute if the element is supported.
- It MAY have a `type` attribute of type String that distinguishes it within a general resource class. A compliant implementation SHOULD support this attribute if the element is supported.

The following are two examples for specifying a Resource element:

```
<Resource name="License" type="MATLAB">1</Resource>
<Resource name="Telescope" type="Zoom2000" duration="750" metric="KX">10</Resource>
```

Extension Element

The Extension element provides a means to pass extensible properties with the job object.

Some applications might find it easier to use a named extension property than discover and handle elements they do not understand or anticipate by name:

- A compliant implementation MAY support this element.
- This element MUST have a `name` attribute of type String that gives the extension property's name. A compliant implementation MUST support this attribute if this element is supported.
- This element MAY have a `type` attribute of type String that characterizes the context within which the property should be understood. A compliant implementation SHOULD support this attribute if this element is supported.
- This element's character content, which is of type String, is the extension property's value.

The following is an example of an Extension element:

```
<Extension type="Scheduler" name="Restartable">true</Extension>
```

TaskGroup

A job MAY specify one or more task groups.

See the next section for element details.

TaskGroupDefaults

A job MAY specify zero or more task group defaults.

See the next section for element details.

5.2 Job Reference

When a simple reference to a predefined job is needed in an encapsulating element, a Job element is used with the text content being the job ID:

```
<Job> job123</Job>
```

6.0 TaskGroup and TaskGroupDefaults Element

The `TaskGroup` and `TaskGroupDefaults` elements have the same structure. A `TaskGroup` element aggregates tasks. A TaskGroupDefaults element can only appear within a Job (or JobDefaults) and represents the defaults to be taken by all task groups within the job. Task group properties in `TaskGroup` elements override any properties found in a sibling `TaskGroupDefaults` element.

- A compliant implementation MAY support the `TaskGroup` element.
- A compliant implementation MAY support the `TaskGroupDefaults` element.
- A task group MUST specify one or more TaskGroup Properties.
- One or more Task elements MAY appear at this level.
- Zero or one TaskDefaults elements MAY appear at this level.

The following illustrates this element's syntax:

```
<TaskGroup>
  <!-- TaskGroup Properties -->+
  <!-- Job Properties -->*
  <Task>+
  <TaskDefaults>?
</TaskGroup>
```

6.1 TaskGroup Properties

TaskGroup Properties apply to a particular task group or as default properties to encompassed task groups. These properties include the task group ID, its tasks, task defaults, and other simple task group properties.

Simple TaskGroup Properties

Simple (unstructured) task group properties are enumerated in the table below:

Table M-5: Simple TaskGroup Properties

Element Name	Type	Description	Appearance	Compliance
TaskCount	Integer	Number of tasks in this task group.	MAY	MUST
Id	String	A task group identifier unique within the job.	MAY	MAY
Name	String	A task group name (such as Master).	MAY	SHOULD

Task

A task group MAY specify zero or more tasks.

See the next section for element details.

TaskDefaults

A task group MAY specify zero or more task defaults.

See the next section for element details.

6.2 TaskGroup Reference

When a simple reference to a predefined task group is needed in an encapsulating element, a TaskGroup element is used with the text content being the task group ID:

```
<TaskGroup> tg1</TaskGroup>
```

7.0 Task and TaskDefaults Element

The `Task` and `TaskDefaults` elements have the same structure. A `Task` element contains information specific to a task (like the process ID or the host it ran on). A `TaskDefaults` element can only appear within a `TaskGroup` (or `TaskGroupDefaults`) element and represents the defaults for all tasks within the task group. Task properties in `Task` elements override any properties found in a sibling `TaskDefaults` element.

- A compliant implementation MAY support the `TaskGroup` element.
- A compliant implementation MAY support the `TaskGroupDefaults` element.
- A task group MUST specify one or more `TaskGroup` Properties.

- One or more `Task` elements MAY appear at this level.
- Zero or one `TaskDefaults` elements MAY appear at this level.

The following illustrates this element's syntax:

```
<Task>
  <!-- Task Properties -->+
  <!-- Job Properties -->*
</Task>
```

7.1 Task Properties

Task Properties are properties that apply to a particular task or as default properties to encompassed tasks. These properties include the task ID and other task properties.

Simple Task Properties

Simple (unstructured) task properties are enumerated in the table below:

Table M-6: Simple Task Properties

Element Name	Type	Description	Appearance	Compliance
Node	String	Name of the node this task ran on.	MAY	MUST
Session	Integer	Session ID for the task group or job.	MAY	MAY
Id	String	A task identifier unique within the task group.	MAY	MAY

7.2 Task Reference

When a simple reference to a predefined task is needed in an encapsulating element, a `Task` element is used with the text content being the task ID:

```
<Task>1</Task>
```

8.0 Property Categories

Certain properties need to be classified as being in a particular category. This is done when it is necessary to distinguish between a property that is requested and a property that was delivered. When no such distinction is necessary, we recommend that the property not be enveloped in one of these elements. In general, a property should be enveloped in a category element only if it is expected that the property will need to be attributed to more than one property category, or if it needs to make use of some of the special attributes inherited from the category.

8.1 Requested Element

A requested property reflects properties as they were requested. A disparity might occur between the requested value and the value delivered if a preference was expressed, if multiple options were specified, or if ranges or pattern matching was specified.

- A compliant implementation SHOULD support this element.

The following illustrates the syntax of this element:

```
<Requested>
  <!-- Requested Properties -->+
</Requested>
```

The following describes the attributes and elements for the example above:

`/Requested`

This element is used to encapsulate requested properties:

`/Requested/<Requested Property>`

Requested properties appear at this level.

Requested Properties inherit some additional attributes:

- A requested property MAY have an `op` attribute of type String that indicates a conditional operation on the value. A compliant implementation SHOULD support this attribute. Valid values for the `op` attribute include EQ meaning equals (which is the default), NE meaning not equal, LT meaning less than, GT meaning greater than, LE meaning less than or equal to, GE meaning greater than or equal to, Match that implies the value is a pattern to be matched.
- A requested property MAY have a `conj` attribute of type String that indicates a conjunctive relationship with the previous element. A compliant implementation MAY support this attribute. Valid values for the `conj` attribute include And (which is the default), Or, Nand meaning and not, and Nor meaning or not.
- A requested property MAY have a `group` attribute of type Integer that indicates expression grouping and operator precedence much like parenthetical groupings. A compliant implementation MAY support this attribute. A positive grouping indicates the number of nested expressions being opened with the property while a negative grouping indicates the number of nested expressions being closed with the property.

- A requested property MAY have a `preference` attribute of type Integer that indicates a preference for the property along with a weight (the weights are taken as a ratio to the sum of all weights in the same group). A compliant implementation MAY support this attribute. If a group of positive valued preference alternatives are specified, at least one of the preferences must be satisfied for the job to run. If a group of negative valued preferences are specified, the preferences will try to be met according to their weights but the job will still run even if it can't satisfy any of the preferred properties. (Weight ranking can be removed by making all weights the same value (1 or -1 for example).
- A requested property MAY have a `performanceFactor` attribute of type Float that provides a hint to the scheduler of what performance tradeoffs to make in terms of resources and start time. A compliant implementation MAY support this attribute.

The following are four examples of using Requested Properties:

```
<Requested>
  <Processors op=”GE”>8</Processors>
  <Processors op=”LE”>16</Processors>
  <Duration>3600</Duration>
</Requested>
<Requested>
  <NodeCount>1</NodeCount>
  <Node aggregation=”Pattern”>fr15.*</Node>
<Requested>
<Requested>
  <User group=”1”>scottmo</User>
  <Account group=”-1”>mscfops</Account>
  <User conj=”Or” group=”1”>amy</User>
  <Account group=”-1”>chemistry</Account>
</Requested>
<Requested>
  <Memory preference=”2”>1024</Memory>
  <Memory preference=”1”>512</Memory>
</Requested>
```

8.2 Delivered Element

A delivered property reflects properties as they were actually utilized, realized or consumed. It reflects the actual amounts or values that are used, as opposed to a limit, choice or pattern as may be the case with a requested property.

- A compliant implementation SHOULD support this element.

The following illustrates the syntax of this element:

```
<Delivered>
  <!-- Delivered Properties -->+
</Delivered>
```

The following describes the attributes and elements for the example above:

`/Delivered`

This element is used to encapsulate delivered properties:

`/Delivered/<Delivered Property>`

Delivered properties appear at this level.

Delivered Properties inherit some additional attributes:

- A delivered property MAY have a `group` attribute of type Integer that indicates expression grouping and operator precedence much like parenthetical groupings. A compliant implementation MAY support this attribute. A positive grouping indicates the number of nested expressions being opened with the property while a negative grouping indicates the number of nested expressions being closed with the property. The purpose of this attribute is to logically group delivered properties if they were used in certain aggregations (like a job that spanned machines).

The following are the same four examples distinguishing the delivered amounts and values:

```
<Delivered>
  <Processors>12</Processors>
  <Duration>1234</Duration>
</Delivered>
<Delivered>
  <Node>fr15n03</Node>
</Delivered>
<Delivered>
  <User>scottmo</User>
  <Account>mscfops</Account>
</Delivered>
<Delivered>
  <Memory>1024</Memory>
</Delivered>
```

9.0 AwarenessPolicy Attribute

A word or two should be said about compatibility mechanisms. With all the leeway in the specification with regard to implementing various portions of the specification, problems might arise if an implementation simply ignores a portion of a job specification that is critical to the job function in certain contexts. Given this situation, it might be desirable in some circumstances for jobs to be rejected by sites that fail to fully support that job's element or attributes. At other times, it might be desirable for a job to run, using a best-effort approach to supporting unimplemented features. Consequently, we define an `awarenessPolicy` attribute that can be added as an optional attribute to the Job element or any other containment or property element to indicate how the property (or the default action for the elements that the containment element encloses) must react when the implementation does not understand an element or attribute.

An awareness policy of `Reject` will cause the server to return a failure if it receives a client request where it does not support an associated element name or attribute name or value. It is reasonable for an implementation to ignore (not even look for) an element or attribute that would not be critical to its function as long as ignoring this attribute or element would not cause an incorrect result. However, any element or attribute that was present that would be expected to be handled in a manner that the implementation does not support must result in a failure.

An awareness policy of `Warn` will accept the misunderstood element or attribute and continue to process the job object on a best effort basis. However a warning MUST be sent (if possible) to the requestor enumerating the elements and attributes that are not understood.

An awareness policy of `Ignore` will accept the unsupported element or attribute and continue to process the job object on a best effort basis. The action could be to simply ignore the attribute.

- The name of this attribute is `awarenessPolicy`.
- This attribute is of type String.
- This attribute can have values of `Reject`, `Warn` or `Ignore`.
- A compliant implementation MAY support this attribute.
- An implementation that does not support an attribute MUST reject any job object that contains elements or attributes that it does not support. Furthermore, it SHOULD return a message to the requestor with an indication of the element or attribute name it did not understand.
- This attribute MAY be present in a property or containment element.
- If an implementation does support the attribute, but it is absent, the default value of `Reject` is implied.
- Individual elements in the job object may override the containing object's awareness policy default by including this attribute. For example, a job might specify an `awarenessPolicy` of Reject at its root (the Job element) but may want to allow a particular subset of elements or attributes to be ignored if not understood. Conversely, a job with a default awarenessPolicy of Ignore might want to classify a subset of its optional elements as Reject if they are indispensable to its correct interpretation. An implementation can opt to check or not check for this attribute at any level it wants but must assume a Reject policy for any elements it does not check.

10.0 References

ISO 8601

ISO (International Organization for Standardization). Representations of dates and times, 1988-06-15. https://www.iso.ch/markete/8601.pdf

DATATYPES

XML Schema Part 2: Datatypes. Recommendation, 02 MAY 2001. https://www.w3.org/TR/xmlschema-2/

11.0 Units of Measure Abbreviations

Abbreviation	Definition	Quantity
B	byte	1 byte
KB	Kilobyte	2^10 bytes
MB	Megabyte	2^20 bytes
GB	Gigabyte	2^30 bytes
TB	Terabyte	2^40 bytes
PB	Petabyte	2^50 bytes
EB	Exabyte	2^60 bytes
ZB	Zettabyte	2^70 bytes
YB	Yottabyte	2^80 bytes
NB	Nonabyte	2^90 bytes
DB	Doggabyte	2^100 bytes

M.2 Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Message Format

Resource Management Interface Specs
Release Version 3.0.4
18 July 2005

Scott Jackson
Brett Bode
David Jackson
Kevin Walker

Status of this Memo

This is a specification defining an XML message format used between Scalable Systems Software components. It is intended that this specification will continue to evolve as these interfaces are implemented and thoroughly tested by time and experience.

Abstract

This document is a specification describing a message format for the interaction of resource management and accounting software components developed as part of the Scalable Systems Software Center. The SSSRMAP Message Format defines a request-response syntax supporting both functional and object-oriented messages. The protocol is specified in XML Schema Definition. The message elements defined in this specification are intended to be framed within the Envelope and Body elements defined in the SSSRMAP Wire Protocol specification document.

Table of Contents

1.0 Introduction

A major objective of the Scalable Systems Software [SSS] Center is to create a scalable and modular infrastructure for resource management and accounting on terascale clusters including resource scheduling, grid-scheduling, node daemon support, comprehensive usage accounting and user interfaces emphasizing portability to terascale vendor operating systems. Existing resource management and accounting components feature disparate

APIs (Application Programming Interfaces) requiring various forms of application coding to interact with other components.

This document proposes a common message format expressed in an XML request-response syntax to be considered as the foundation of a standard for communications between and among resource management and accounting software components. In this document this standard is expressed in two levels of generality. The features of the core SSSRMAP protocol common to all resource management and accounting components in general are described in the main body of this document. The aspects of the syntax specific to individual components are described in component-specific binding documents.

2.0 Conventions Used in this Document

2.1 Keywords

The keywords 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in RFC2119 [RFC2119].

2.2 XML Case Conventions

In order to enforce a consistent capitalization and naming convention across all SSSRMAP specifications 'Upper Camel Case' (UCC) and 'Lower Camel Case' (LCC) Capitalization styles shall be used. UCC style capitalizes the first character of each word and compounds the name. LCC style capitalizes the first character of each word except the first word. [XML_CONV][FED_XML]

1. SSSRMAP XML Schema and XML instance documents SHALL use the following conventions:
    - Element names SHALL be in UCC convention (example: <UpperCamelCaseElement/>).
    - Attribute names SHALL be in LCC convention (example: <UpperCamelCaseElement lowerCamelCaseAttribute=”Whatever”/>).
2. General rules for all names are:
    - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
    - Underscores (_), periods (.) and dashes (−) MUST NOT be used (example: use JobId instead of JOB.ID, Job_ID or job-id).

2.3 Schema Definitions

```
SSSRMAP Schema Definitions appear like this
```

In case of disagreement between the schema file and this specification, the schema file takes precedence.

3.0 Encoding

Encoding tells how a message is represented when exchanged. SSSRMAP data exchange messages SHALL be defined in terms of XML schema [XML_SCHEMA].

3.1 Schema Header and Namespaces

The header of the schema definition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
  xmlns="https://www.w3.org/2001/XMLSchema"
  xmlns:sssrmap="https://scidac.org/ScalableSystems/SSSRMAP"
  targetNamespace="https://www.scidac.org/ScalableSystems/SSSRMAP"
  elementFormDefault="qualified">
```

3.2 Element Descriptions

The following subsections describe the elements that make up SSSRMAP messages. SSSRMAP messages are transmitted in the Body and Envelope elements as described in the SSSRMAP Wire Protocol specification [WIRE_PROTOCOL].

The Request Element

The `Request` element specifies an individual request. An object-oriented request will have at least one `Object` element while a functional request will not have one. Depending on context, the `Request` element MAY contain one or more `Get` elements or one or more Set elements and any number of `Where` elements. `Option`, `Data`, `File` or `Count` elements may also be included. If a component supports it, chunking may be requested where large response data is possible. Setting the chunking attribute to 'True' requests that the server break a large response into multiple chunks (each with their own envelope) so they can be processed in separate pieces.

Only an `action` attribute is required. All other attributes are optional.

Attribute	Description
action	Specifies the action or function to be performed.
actor	The authenticated user sending the request.

Attribute	Description
id	Uniquely maps the request to the appropriate response.
chunking	Requests that segmentation be used for large response data if set to 'True'.
chunkSize	Requests that the segmentation size be no larger than the specified amount.

```
<complexType name="RequestType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="sssrmap:Object" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Option" minOccurs="0" maxOccurs="unbounded"/>
    <choice minOccurs="0" maxOccurs="1">
      <element ref="sssrmap:Get" minOccurs="1" maxOccurs="unbounded"/>
      <element ref="sssrmap:Set" minOccurs="1" maxOccurs="unbounded"/>
    </choice>
    <element ref="sssrmap:Where" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Data" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:Count" minOccurs="0" maxOccurs="1"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </choice>
  <attribute name="action" type="string" use="required"/>
  <attribute name="actor" type="string" use="required"/>
  <attribute name="id" type="string" use="optional"/>
  <attribute name="chunking" type="sssrmap:BoolType" use="optional"/>
  <attribute name="chunkSize" type="positiveInteger" use="optional"/>
</complexType>

<element name="Request" type="sssrmap:RequestType"/>
```

The Object Element

The `Object` element is used in an object-oriented request to specify the object receiving the action. It is possible to have multiple `Object` elements in a request if an implementation supports multi-object queries.

The object class name is specified as text content. All attributes are optional.

- `join` – the type of join to be performed with the preceding object
    - A `join` attribute of 'Inner' specifies an inner join. This is the default.
    - A `join` attribute of 'FullOuter' specifies a full outer join.
    - A `join` attribute of 'LeftOuter' specifies a left outer join.
    - A `join` attribute of 'RightOuter' specifies a right outer join.
    - A `join` attribute of 'Cross' specifies a cross join.
    - A `join` attribute of 'Union' specifies a union join.

```
<complexType name="ObjectType">
  <simpleContent>
    <extension base="string">
      <attribute name="join" type="string" use="optional"/>
    </extension>
  </simpleContent>
```

```
</complexType>

<element name="Object" type="sssrmap:ObjectType"/>
```

The Get Element

The `Get` element is used to indicate the data fields to be returned in a query. Get is typically used within requests with `action="query"`. Multiple Get elements cause the fields to be returned in the order specified. If no `Get` elements are specified, the query will return a default set of fields.

Only a `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the data field to be returned. This MUST be of the form of a 'Modified XPATH expression' as described in a later section.
op	The operator to be used to aggregate or perform an operation on the returned values: • An *op* attribute of 'Sort' specifies an ascending sort operation. • An *op* attribute of 'Tros' specifies a descending sort operation. • An *op* attribute of 'Sum' returns the sum (only valid for numeric values). • An *op* attribute of 'Max' returns the maximum value. • An *op* attribute of 'Min' returns the minimum value. • An *op* attribute of 'Count' returns the number of values. • An *op* attribute of 'Average' returns the average of the values. • An *op* attribute of 'GroupBy' signifies that aggregates are grouped by this field.
object	Specifies the object for which you want the named attribute in a multi-object query.
units	The units in which to return the value (if applicable).

```
<complexType name="GetType">
  <attribute name="name" type="string" use="required"/>
  <attribute name="object" type="string" use="optional"/>
  <attribute name="op" type="sssrmap:GetOperatorType" use="optional"/>
  <attribute name="units" type="string" use="optional"/>
</complexType>

<element name="Get" type="sssrmap:GetType"/>

<simpleType name="GetOperatorType">
  <restriction base="string">
    <enumeration value="Sort"/>
    <enumeration value="Tros"/>
    <enumeration value="Count"/>
    <enumeration value="Sum"/>
    <enumeration value="Max"/>
    <enumeration value="Min"/>
    <enumeration value="Average"/>
    <enumeration value="GroupBy"/>
  </restriction>
</simpleType>
```

The Set Element

The `Set` element is used to specify the object data fields to be assigned values. Set is typically used within requests with `action=”Create”` or `action=”Modify”`. The use of `Get` or `Set` elements within a request is mutually exclusive.

The assignment value (to which the field is being changed) is specified as the text content. A Set element without a value can be used as an assertion flag. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the field being assigned a value. This MUST be of the form of a 'Modified XPATH expression' as described in a later section.
op	The operator to be used in assigning a new value to the name. If an `op` attribute is not specified and a value is specified, the specified value will be assigned to the named field ('assign'). • An `op` attribute of 'Assign' assigns value to the named field. • An `op` attribute of 'Inc' increments the named field by the value. • An `op` attribute of 'Dec' decrements the named field by the value.
units	The units corresponding to the value being set.

```
<complexType name="SetType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:SetOperatorType" use="optional"/>
      <attribute name="units" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Set" type="sssrmap:SetType"/>

<simpleType name="SetOperatorType">
  <restriction base="string">
    <enumeration value="Assign"/>
    <enumeration value="Inc"/>
    <enumeration value="Dec"/>
  </restriction>
</simpleType>
```

The Where Element

A `Request` element can contain one or more `Where` elements that specify the search conditions for which objects the action is to be performed on.

The condition value (against which the field is tested) is specified as the text content. A `Where` element without a value can be used as a truth test. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the data field to be tested. This MUST be of the form of a 'Modified XPATH expression' as described in a later section.
op	The operator to be used to test the name against the value. If an op attribute is not specified and a value is specified, the field will be tested whether it is equal to the value ('EQ'): • An `op` attribute of 'EQ' specifies an equality comparison. • An `op` attribute of 'LT' specifies a 'less than' comparison. • An `op` attribute of 'GT' specifies a 'greater than' comparison. • An `op` attribute of 'LE' specifies a 'less than or equal to' test. • An `op` attribute of 'GE' specifies a 'greater than or equal to' test. • An `op` attribute of 'NE' specifies a 'not equal to' test. • An `op` attribute of 'Match' specifies a regular expression matching comparison.
conj	Indicates whether this test is to be ANDed or ORed with the immediately preceding where condition: • A `conj` attribute of 'And' specifies an 'and' conjunction. • A `conj` attribute of 'Or' specifies an 'or' condition. • A `conj` attribute of 'AndNot' specifies an 'and not' conjunction. • A `conj` attribute of 'OrNot' specifies an 'or not' condition.
group	Indicates an increase or decrease of parentheses grouping depth: • A positive number indicates the number of left parentheses to precede the condition [i.e., `group=”2”` represents “((condition”]. • A negative number indicates the number of right parentheses to follow the condition [i.e., `group=”-2”` represents “condition))”].
object	Specifies the object for the first operand in a multi-object query.
subject	Specifies the object for the second operand in a multi-object query.
units	Indicates the units to be used in the value comparison.

```
<complexType name="WhereType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:OperatorType" use="optional"/>
      <attribute name="conj" type="sssrmap:ConjunctionType" use="optional"/>
      <attribute name="group" type="integer" use="optional"/>
      <attribute name="units" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="Where" type="sssrmap:WhereType"/>

<simpleType name="WhereOperatorType">
  <restriction base="string">
    <enumeration value="EQ"/>
    <enumeration value="GT"/>
    <enumeration value="LT"/>
    <enumeration value="GE"/>
    <enumeration value="LE"/>
    <enumeration value="NE"/>
    <enumeration value="Match"/>
  </restriction>
</simpleType>
```

The Option Element

The `Option` element is used to indicate processing options for the command. An option might be used to indicate that command usage or special formatting is desired, or that the command is to be invoked with particular options.

The option value is specified as the text content. An `Option` element without a value can be used as an assertion flag. Only the `name` attribute is required. All other attributes are optional.

Attribute	Description
name	The name of the field being assigned a value.
op	The operator to be used to disassert the option: • An `op` attribute of 'Not' specifies that the option is not asserted.
conj	Indicates whether this test is to be ANDed or ORed with the immediately preceding where condition: • A `conj` attribute of 'And' specifies an 'and' conjunction. • A `conj` attribute of 'Or' specifies an 'or' condition. • A `conj` attribute of 'AndNot' specifies an 'and not' conjunction. • A `conj` attribute of 'OrNot' specifies an 'or not' condition.

```
<complexType name="OptionType">
  <simpleContent>
    <extension base="string">
      <attribute name="name" type="string" use="required"/>
      <attribute name="op" type="sssrmap:OptionOperatorType" use="optional"/>
      <attribute name="conj" type="sssrmap:ConjunctionType" use="optional"/>
    </extension>
  </simpleContent>
</complexType>
```

```
<element name="Option" type="sssrmap:OptionType"/>

<simpleType name="OptionOperatorType">
  <restriction base="string">
    <enumeration value="Not"/>
  </restriction>
</simpleType>
```

The Data Element

A `Request` or `Response` element can have one or more `Data` elements that allow the supplying of context-specific data. A request might pass in a structured object via a `Data` element to be acted upon. Typically a query will result in a response with the data encapsulated within a `Data` element.

The following attributes are optional:

Attribute	Description
name	Object name describing the contents of the data.
type	Describing the form in which the data is represented: • A `type` attribute of 'XML' indicates the data has internal xml structure and can be recursively parsed by an XML parser. • A `type` attribute of 'Binary' indicates the data is an opaque dataset consisting of binary data. • A `type` attribute of 'String' indicates the data is an ASCII string. • A `type` attribute of 'Int' indicates the data is an integer. • A `type` attribute of 'Text' indicates the data is in formatted human-readable text. • A `type` attribute of 'HTML' indicates the data is represented in HTML.

```
<complexType name="DataType">
  <sequence>
   <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="name" type="string" use="optional"/>
  <attribute ref="sssrmap:Type" use="optional"/>
</complexType>

<element name="data" type="sssrmap:DataType"/>
```

The File Element

A `Request` or `Response` element can have one or more `File` elements of type String that allow the inclusion of files. The files can be either text or binary and can be referenced by objects inside the Data element. A file can be compressed using the gzip algorithm [ZIP]. A binary file or a compressed file must be base64 encoded as defined in XML Digital Signatures (`https://www.w3.org/2000/09/xmldsig#base64`). Metadata describing the modes and properties of the resulting file are passed as parameters. The text or base64 encoded file data forms the string content of the `File` element.

The following attributes are optional:

Attribute	Description
id	Specifies an identifier that allows the file to be referenced from within another object. If more than one File elements are specified, this attribute is REQUIRED in each of them.
name	Specifies the name to give the file upon creation on the target system. This can be an absolute or relative pathname (relative to the InitialWorkingDirectory).
owner	Indicates what owner the file should be changed to. By default, it will be changed to the UserId that the authenticated actor maps to on the target system. Note that this function should succeed only if the requestor has the privileges to do so (i.e., authenticated as root).
group	Indicates what group the file should be changed to. By default, it will be set to the primary groupid of the UserId that the authenticated actor maps to on the target system. Note that this function should succeed only if the requestor has the proper privileges.
mode	Indicates the permissions the file should possess. By default, it will be set according to the default umask for the UserId that the authenticated actor maps to on the target system. Note that this function should not set permissions for the file that exceed the privileges for the actor. These permissions can be specified using either an octal number or symbolic operations (as accepted by the GNU chmod(1) command).
compressed	Indicates whether the file has been compressed: • A `compressed` attribute of 'True' indicates the file has been compressed. • A `compressed` attribute of 'False' indicates the file has not been compressed. This is the default.
encoded	Indicates whether the file has been base64 encoded: • An `encoded` attribute of 'True' indicates the file has been encoded. • An `encoded` attribute of 'False' indicates the file has not been encoded. This is the default.

```
<complexType name="FileType">
  <sequence>
    <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="name" type="string" use="optional"/>
  <attribute name="owner" type=="string" use="optional"/>
  <attribute name="group" type="string" use="optional"/>
  <attribute name="mode" type="string" use="optional"/>
  <attribute name="compressed" type="boolean" use="optional"/>
  <attribute name="encoded" type="boolean" use="optional"/>
</complexType>

<element name="file type="sssrmap:FileType"/>
```

The Count Element

A single `Count` element can be included within a `Request` or `Response` and is context-specific. This can be used to represent the number of objects acted upon or returned.

```
<element name="Count" type="positiveInteger"/>
```

The Response Element

The `Response` element specifies an individual response. It MUST contain a `Status` element. It MAY also contain `Count` and any number of `Data` or `File` elements. If chunking has been requested and is supported by the server, a large response can be broken up into multiple chunks (each with their own envelope). The `chunkNum` attribute can be used to indicate which chunk the current one is. The `chunkMax` attribute can be used to determine when all the chunks have been received (all chunks have been received if `chunkNum=chunkMax` or `chunkMax=0`).

It MAY have any of the following attributes:

Attribute	Description
id	Uniquely maps the response to the corresponding request.
chunkNum	Integer indicating the current chunk number [1 is implied when this attribute is missing or blank].
chunkMax	Integer indicating the number of chunks expected [-1 means unknown but more chunks to follow; 0 means unknown but this is the last chunk; 0 is implied if this attribute is missing or blank].

```
<complexType name="ResponseType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="sssrmap:Status" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Count" minOccurs="0" maxOccurs="1"/>
    <element ref="sssrmap:Data" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="sssrmap:File" minOccurs="0" maxOccurs="unbounded"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other"/>
  </choice>
  <attribute name="object" type="string" use="optional"/>
  <attribute name="action" type="string" use="optional"/>
  <attribute name="id" type="string" use="optional"/>
  <attribute name="chunkNum" type="integer" use="optional"/>
  <attribute name="chunkMax" type="integer" use="optional"/>
</complexType>

<element name="Response" type="sssrmap:ResponseType"/>
```

The Status Element

A `Response` element MUST contain a single `Status` element that indicates whether the reply represents a success, warning or failure. This element is composed of the child elements `Value`, `Code` and `Message`. Of these, `Value` and `Code` are required, and `Message` is optional.

```
<complexType name="StatusType">
  <choice minOccurs="1" maxOccurs="unbounded">
    <element ref="sssrmap:Value" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Code" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:Message" minOccurs="0" maxOccurs="1"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other"/>
  </choice>
</complexType>

<element name="Status" type="sssrmap:StatusType"/>
```

The Value Element

The `Value` element is of type String and MUST have a value of 'Success', 'Warning' or 'Failure':

```
<simpleType name="StatusValueType">
  <restriction base="string">
    <enumeration value="Success"/>
    <enumeration value="Warning"/>
    <enumeration value="Failure"/>
  </restriction>
</simpleType>

<element name="Value" type="sssrmap:StatusValueType"/>
```

The Code Element

A `Response` element must contain a single `Code` element that specifies the 3-digit status code for the response. Refer to the next section on Error Reporting for a description and listing of supported status codes.

```
<simpleType name="CodeType">
  <restriction base="string">
    <pattern value="[0-9]{3}"/>
  </restriction>
</simpleType>

<element name="Code" type="sssrmap:CodeType"/>
```

The Message Element

A `Response` element can contain a single `Message` element that is context specific to the success or failure response. The message should be an error message if status is false. If present for a successful response, it can be used as a human readable message for a user interface.

```
<element name="Message" type="string"/>
```

3.3 Modified XPATH Expressions

The `name` attribute used within the `Get`, `Set` and `Where` Elements MUST have the form of a modified XPATH expression as defined in this section. Usually this will just be the simple name of the object property. Some complex objects, such as the SSS Job Object and the SSS Node Object, however, are represented in a structured way with nested elements. In order to define a consistent and flexible way to access and manipulate these objects and keep the flat XML objects simple and straightforward, SSSRMAP specifies that a 'Modified XPATH' syntax be used.

In essence, 'Modified XPATH' is defined to be an XPATH [XPATH] expression with the exception that the '//' can be omitted from the beginning of the expression when a document search is desired. Therefore, on the server side, a standard XPATH routine can be used by prepending '//' to any expression that does not begin with a '/'.

The response data should always include all of the structure of the queried object necessary to place the requested data in its proper context.

See the XPATH specification for a full description of XPATH. The XPath 1.0 Recommendation is `https://www.w3.org/TR/1999/REC-xpath-19991116`. The latest version of XPath 1.0 is available at xpath cover page.

Sample Modified XPATH Expressions

Consider the following hypothetical object(s) (which might be returned within a Data element):

```
<Job>
  <JobId>PBS.1234.0</JobId>
  <Requested>
    <Memory op=”GE”>512</Memory>
    <Processors>2</Processors>
    <WallDuration>P3600S</WallDuration>
  </Requested>
  <Utilized>
    <Memory metric=”Average”>488</Memory>
    <WallDuration>P1441S</WallDuration>
  </Utilized>
</Job>
```

To get everything above for this job you do not need a `Get` element:

```
<Request action=”Query”>
  <Object>Job</Object>
  <Where name=”JobId”>PBS.1234.0</Where>
</Request>
```

If you used `<Get name=”JobId”/>` you would get back:

```
<Job>
  <JobId>PBS.1234.0</JobId>
</Job>
```

If you used `<Get name="Memory"/>` (or `name="/Job/*/Memory"`) you would get:

```
<Job>
  <Requested>
    <Memory op="GE">512</Memory>
  </Requested>
  <Utilized>
    <Memory metric="Average">488</Memory>
  </Utilized>
</Job>
```

If you used `<Get name="Requested/Memory"/>` (or `name="/Job/Requested/Memory"`) you would get:

```
<Job>
  <Requested>
    <Memory op="GE">512</Memory>
  </Requested>
</Job>
```

If you used `<Get name="Memory[@metric='Average']"/>` (or `name="Memory[@metric]"`) you would get:

```
<Job>
  <Utilized>
    <Memory metric="Average">488</Memory>
  </Utilized>
</Job>
```

3.4 Examples

Sample Requests

Requesting a list of nodes with a certain configured memory threshold (batch format):

```
<Request action="Query" id="1">
  <Object>Node</Object>
  <Get name="Name" />
  <Get name="Configured/Memory" />
  <Where name="Configured/Memory" op="GE" units="MB">512</Where>
</Request>
```

Activating a couple of users:

```
<Request action="Modify">
  <Object>User</Object>
  <Set name="Active">True</Set>
  <Where name="Name">scott</Where>
  <Where name="Name" conj="Or"/>brett</Where>
</Request>
```

Submitting a simple job:

```
<Request action="Submit">
  <Object>Job</Object>
  <Data>
    <Job>
      <User>xdp</User>
      <Account>youraccount</Account>
      <Command>myprogram</Command>
      <InitialWorkingDirectory>/usr/home/scl/xdp</InitialWorkingDirectory>
      <RequestedNodes>4</RequestedNodes>
      <RequestedWCTime>100</RequestedWCTime>
    </Job>
  </Data>
</Request>
```

Sample Responses

A response to the available memory nodes query (batch format):

```
<Response id="1">
  <Status>
    <Value>Success</Value>
    <Code>000</Code>
  </Status>
  <Count>2</Count>
  <Data>
    <Node>
      <Name>fr01n01</Name>
      <Configured>
        <Memory>512</Memory>
      </Configured>
    </Node>
    <Node>
      <Name>fr12n04</Name>
      <Configured>
        <Memory>1024</Memory>
      </Configured>
    </Node>
  </Data>
</Response>
```

Two users successfully activated:

```
<Response>
  <Status>
    <Code>000</Code>
    <Message>Two users were successfully modified</Message>
  </Status>
  <Count>2</Count>
</Response>
```

A failed job submission:

```
<Response>
  <Status>
    <Value>Failure</Value>
    <Code>711</Code>
    <Message>Invalid account specified. The job was not submitted.</Message>
  </Status>
</Response>
```

4.0 Error Reporting

SSSRMAP requests will return a status and a 3-digit response code to signify success or failure conditions. When a `request` is successful, a corresponding `response` is returned with the `status` element set to Success and the `code` element set to '000'. When a `request` results in an error detected by the server, a `response` is returned with the `status` element set to Failure and a 3-digit error code in the `code` element. An optional human-readable `message` can also be include in a failure response providing context-specific detail about the failure. The default message language is US English. (The status flag makes it easy to signal success or failure and allows the receiving peer some freedom in the amount of parsing it wants to do on failure [BXXP]).

Success Codes

Code	Response Text in US English
0xx	Request was successful
000	General Success
010	Help/usage reply
020	Status reply
030	Subscription successful
035	Notification successful (Ack)
040	Registration successful
050-079	Component-defined
080-099	Application-defined

Warning Codes

Code	Response Text in US English
1xx	Request was successful but includes a warning

Code	Response Text in US English
100	General warning (examine message for details)
102	Check result (Did what you asked but may not have been what you intended -- or information is suspect)
110	Wire Protocol or Network warning
112	Redirect
114	Protocol warning (something was wrong with the protocol, but best effort guesses were applied to fulfill the request)
120	Message Format warning
122	Incomplete specification (request missing some essential information -- best effort guess applied)
124	Format warning (something was wrong with the format but best effort guesses were applied to fulfill the request)
130	Security warning
132	Insecure request
134	Insufficient privileges (Response was sanitized or reduced in scope due to lack of privileges)
140	Content or action warning
142	No content (The server has processed the request but there is no data to be returned)
144	No action taken (nothing acted upon -- i.e., deletion request did not match any objects)
146	Partial content
148	Partial action taken
150-179	Component-defined
180-199	Application-defined

Wire Protocol Codes

Code	Response Text in US English
2xx	A problem occurred in the wire protocol or network
200	General wire protocol or network error
210	Network failure
212	Cannot resolve host name
214	Cannot resolve service port
216	Cannot create socket
218	Cannot bind socket
220	Connection failure
222	Cannot connect
224	Cannot send data
226	Cannot receive data
230	Connection rejected
232	Timed out
234	Too busy
236	Message too large
240	Framing failure
242	Malformed framing protocol
244	Invalid payload size
246	Unexpected end of file
250-279	Component-defined

Code	Response Text in US English
280-299	Application-defined

Message Format Codes

Code	Response Text in US English
3xx	A problem occurred in the message format
300	General message format error
302	Malformed XML document
304	Validation error (XML Schema)
306	Namespace error
308	Invalid message type (Something other than Request or Response in Body)
310	General syntax error in request
311	Object incorrectly (or not) specified
312	Action incorrectly (or not) specified
313	Invalid Action
314	Missing required element or attribute
315	Invalid Object (or Object-Action combination)
316	Invalid element or attribute name
317	Illegal value for element or attribute
318	Illegal combination
319	Malformed Data
320	General syntax error in response

Code	Response Text in US English
321	Status incorrectly (or not) specified
322	Code incorrectly (or not) specified
324	Missing required element or attribute
326	Invalid element or attribute name
327	Illegal value for element or attribute
328	Illegal combination
329	Malformed Data
340	Pipelining failure
342	Request identifier is not unique
344	Multiple messages not supported
346	Mixed messages not supported (both requests and responses in same batch)
348	Request/response count mismatch
350-379	Component-defined
380-399	Application-defined

Security Codes

Code	Response Text in US English
4xx	A security requirement was not fulfilled
400	General security error
410	Negotiation failure
412	Not understood

Code	Response Text in US English
414	Not supported
416	Not accepted
420	Authentication failure
422	Signature failed at client
424	Authentication failed at server
426	Signature failed at server
428	Authentication failed at client
430	Encryption failure
432	Encryption failed at client
434	Decryption failed at server
436	Encryption failed at server
438	Decryption failed at client
440	Authorization failure
442	Authorization failed at client
444	Authorization failed at server
450-479	Component-defined
480-499	Application-defined

Event Management Codes

Code	Response Text in US English
5xx	Failure conditions in event messaging

Code	Response Text in US English
500	General Event Management failure
510	Subscription failed
520	Notification failed
550-579	Component-defined
580-599	Application-defined

Reserved Codes

Code	Response Text in US English
6xx	Reserved for future use

Server Application Codes

Code	Response Text in US English
7xx	A server-side application-specific error occurred
700	General failure
710	Not supported
712	Not understood
720	Internal error
730	Resource unavailable (insufficient resources -- software, hardware or a service I rely upon is down)
740	Business logic
750-779	Component-defined

Code	Response Text in US English
780-799	Application-defined

Client Application Codes

Code	Response Text in US English
8xx	A client-side application-specific error occurred
800	General failure
810	Not supported
812	Not understood
820	Internal error
830	Resource unavailable
840	Business logic
850-879	Component-defined
880-899	Application-defined

Miscellaneous Codes

Code	Response Text in US English
9xx	Miscellaneous failures
999	Unknown failure

5.0 References

[BEEP] M. Rose, “The Blocks Extensible Exchange Protocol Core”, RFC 3080, March 2001.

[FED_XML] “U.S. Federal XML Guidelines”.

[HMAC] H. Krawczyk, M. Bellare, R. Canetti, “HMAC, Keyed-Hashing for Message Authentication”, RFC 2104, February 1997.

[HTTP] “Hypertext Transfer Protocol – HTTP/1.1”, RFC 2616, June 1999.

[RFC2119] S. Bradner, “Key Words for Use in RFCs to Indicate Requirement Levels”, RFC 2119, March 1997.

[RFC3117] M. Rose, “On the Design of Application Protocols”, Informational RFC 3117, November 2001.

[SHA-1] U.S. Department of Commerce/National Institute of Standards and Technology, “Secure Hash Standard”, FIPS PUB 180-1.

[SSS] “Scalable Systems Software”, https://www.scidac.org/ScalableSystems

[WIRE_PROTOCOL] S. Jackson, B. Bode, D. Jackson, K. Walker, “Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol“, SSS Resource Management and Accounting Documents, January 2004.

[XML] Bray, T., et al, “Extensible Markup Language (XML) 1.0 (Second Edition)”, 6 October 2000.

[XML_CONV] “I-X and <I-N-CA> XML Conventions”.

[XML_DSIG] D. Eastlake, J. Reagle Jr., D. Solo, “XML Signature Syntax and Processing”, W3C Recommendation, 12 February 2002.

[XML_ENC] T. Imamura, B. Dillaway, E. Smon, “XML Encryption Syntax and Processing”, W3C Candidate Recommendation, 4 March 2002.

[XML_SCHEMA] D. Beech, M. Maloney, N. Mendelshohn, “XML Schema Part 1: Structures Working Draft”, April 2000.

[XPath 1.0] J. Clark, S. DeRose, “XML Path Language (XPath) Version 1.0”, 16 November 1999.

[XRP] E. Brunner-Williams, A. Damaraju, N. Zhang, “Extensible Registry Protocol (XRP)”, Internet Draft, expired August 2001.

[ZIP] J. Gailly, M. Adler, “The gzip home page”, https://www.gzip.org/

M.3 Scalable Systems Software Node Object Specification

SSS Node Object Specification
Release Version 3.1.0
26 April 2011

Scott Jackson, PNNL
David Jackson, Ames Lab
Brett Bode, Ames Lab

Status of this Memo

This is a specification of the node object to be used by Scalable Systems Software compliant components. It is envisioned for this specification to be used in conjunction with the SSSRMAP protocol with the node object passed in the Data field of Requests and Responses. Queries can be issued to a node-cognizant component in the form of modified XPATH expressions to the Get field to extract specific information from the node object as described in the SSSRMAP protocol.

Abstract

This document describes the syntax and structure of the SSS node object. This node model takes into account various node property categories such as whether it represents a configured, available or utilized property.

Table of Contents

1.0 Introduction

This specification proposes a standard XML representation for a node object for use by the various components in the SSS Resource Management System. This object will be used in multiple contexts and by multiple components. It is anticipated that this object will be passed via the Data Element of SSSRMAP Requests and Responses.

1.1 Goals

There are several goals motivating the design of this representation.

It needs to be inherently flexible. We recognize we will not be able to exhaustively include the ever-changing node properties and capabilities that constantly arise.

The same node object should be used at all stages of its lifecycle. This object needs to distinguish between configured, available and utilized properties of a node.

Its design takes into account the properties and structure required to function in a meta or grid environment. It should eventually include the capability of resolving namespace and locality issues, though the earliest versions will ignore this requirement.

One should not have to make multiple queries to obtain a single piece of information (i.e., there should not be two mutually exclusive ways to represent a node resource).

It needs to support resource metric and unit specifications.

1.2 Examples

Simple Example

This example shows a simple expression of the Node object:

```
<Node>
  <Id>Node64</Id>
  <Configured>
    <Processors>2</Processors>
    <Memory>512</Memory>
  </Configured>
</Node>
```

Elaborate Example

This example shows a more elaborate Node object:

```
<Node>
  <Id>64</Id>
  <Name>Netpipe2</Name>
  <Feature>BigMem</Feature>
  <Feature>NetOC12</Feature>
  <Opsys>AIX</Opsys>
  <Arch>Power4</Arch>
  <Configured>
    <Processors>16</Processors>
    <Memory units=”MB”>512</Memory>
    <Swap>512</Swap>
  </Configured>
  <Available>
    <Processors>7</Processors>
    <Memory metric=”Instantaneous”>143</Memory>
  </Available>
  <Utilized>
    <Processors wallDuration=”3576”>8</Processors>
    <Memory metric=”Average” wallDuration=”3576”>400</Memory>
  </Utilized>
</Node>
```

2.0 Conventions Used in this Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 Table Column Interpretations

In the property tables, the columns are interpreted to have the following meanings:

Property	Description
Element Name	Name of the XML element (xsd:element).
Type	Data type defined by xsd (XML Schema Definition) as: • String — xsd:string (a finite length sequence of printable characters). • Integer — xsd:integer (a signed finite length sequence of decimal digits). • Float — xsd:float (single-precision 32-bit floating point). • Boolean — xsd:boolean (consists of the literals “true” or “false”). • DateTime — xsd:dateTime (discreet time values are represented in ISO 8601 extended format `CCYY-MM-DDThh:mm:ss` where `CC` represents the century, `YY` the year, `MM` the month and `DD` the day. The letter `T` is the date/time separator and `hh`, `mm`, `ss` represent hour, minute and second respectively. This representation may be immediately followed by a `Z` to indicate Coordinated Universal Time (UTC) or, to indicate the time zone (i.e., the difference between the local time and Coordinated Universal Time), immediately followed by a sign, + or −, followed by the difference from UTC). • Duration — xsd:duration (a duration of time is represented in ISO 8601 extended format PnYnMnDTnHnMnS, where nY represents the number of years, nM the number of months, nD the number of days, `T` is the date/time separator, nH the number of hours, nM the number of minutes and nS the number of seconds. The number of seconds can include decimal digits to arbitrary precision).
Description	Brief description of the meaning of the property.
Appearance	Indicates whether the given property has to appear within the parent element. It assumes the following meanings: • MUST — This property is REQUIRED when the parent is specified. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.
Compliance	Indicates whether a compliant implementation has to support the given property: • MUST — A compliant implementation MUST support this property. • SHOULD — A compliant implementation SHOULD support this property. • MAY — A compliant implementation MAY support this property.
Categories	Some properties can be categorized into one of several categories. Letters in this column indicate that the given property can be classified in the following property categories: • C — This property can be encompassed in a Configured element.

Property	Description
	• A — This property can be encompassed in an Available element. • U — This property can be encompassed in a Utilized element.

2.3 Element Syntax Cardinality

The cardinality of elements in the element syntax sections can make use of regular expression wildcards with the following meanings:

Wildcard	Description
*	Zero or more occurrences.
+	One or more occurrences.
?	Zero or one occurrences.

The absence of one of these symbols implies one and only one occurrence.

3.0 The Node Model

The primary element within the node model is a node. One can speak of some node properties as being a configured, available or utilized property of the node.

4.0 Node Element

The Node element is the root element of a node object and is used to encapsulate a node:

- A node object MUST have exactly one Node element.
- A compliant implementation MUST support this element.
- A node MUST specify one or more Node Properties.

4.1 Uncategorized Node Properties

Uncategorized Node Properties are properties that apply to the node as a whole and do not need to be distinguished between being configured, available or utilized. These include the node ID and other optional node properties.

Simple Node Properties

Simple (unstructured) node properties are enumerated in the table below:

Table M-7: Simple Node Properties

Element Name	Type	Description	Appearance	Compliance
Id	String	Node identifier.	MUST	MUST
Name	String	Node name or pattern.	MAY	MAY
OpSys	String	Operating System.	MAY	SHOULD
Arch	String	Architecture.	MAY	SHOULD
Description	String	Description of the node.	MAY	MAY
State	String	State of the node. Valid states can include Offline, Configured, Unknown, Idle, and Busy.	SHOULD	MUST
Features	String	Arbitrary named features of the node (comma-delimited string).	MAY	SHOULD

Extension Element

The `Extension` element provides a means to pass extensible properties with the node object. Some applications might find it easier to deal with a named extension property than discover and handle elements for which they do not understand or anticipate by name.

- A compliant implementation MAY support this element.
- This element MUST have a name attribute that is of type String and represents the name of the extension property. A compliant implementation MUST support this attribute if this element is supported.
- This element MAY have a type attribute that is of type String and provides a hint about the context within which the property should be understood. A compliant implementation SHOULD support this attribute if this element is supported.
- The character content of this element is of type String and is the value of the extension property.

The following is an example of an `Extension` element:

```
<Extension type="Chemistry" name="Software">NWChem</Extension>
```

4.2 Property Categories

Certain node properties (particularly consumable resources) need to be classified as being in a particular category. This is done when it is necessary to distinguish between a property that is configured versus a property that is available or utilized. For example, a node might be configured with 16 processors. At a particular time, 8 might be utilized, 7 might be available and 1 disabled. When a node property must be categorized to be understood properly, the property MUST be enveloped within the appropriate Property Category Element.

Configured Element

A configured node property reflects resources pertaining to the node that could in principle be used though they might not be available at this time. This information could be used to determine if a job could ever conceivably run on a given node.

- A compliant implementation MUST support this element.

The following is an example of using Configured Properties:

```
<Configured>
  <Processors>16</Processors>
  <Memory units=”MB”>512</Memory>
</Configured>
```

Available Element

An available node property refers to a resource that is currently available for use.

- A compliant implementation SHOULD support this element.

The following is an example of specifying available properties:

```
<Available>
  <Processors>7</Processors>
  <Memory units=”MB”>256</Memory>
</Available>
```

Utilized Element

A utilized node property reflects resources that are currently utilized.

- A compliant implementation SHOULD support this element.

The following is an example of specifying utilized properties:

```
<Utilized>
  <Processors>8</Processors>
  <Memory metric=”Average”>207</Memory>
</Utilized>
```

4.3 Categorized Node Properties

Consumable Resources

Consumable Resources are a special group of node properties that can have additional attributes and can be used in multiple categories. In general a consumable resource is a resource that can be consumed in a measurable quantity.

- A consumable resource MUST be categorized as being a configured, available or utilized node property by being a child element of a Configured, Available or Utilized element respectively.
- A consumable resource MAY have a units attribute that is of type String that specifies the units by which it is being measured. If this attribute is omitted, a default unit is implied. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a metric attribute that is of type String that specifies the type of measurement being described. For example, the measurement can be a Total, an Average, a Min or a Max. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a wallDuration attribute of type Duration that indicates the amount of time for which that resource was used. This need only be specified if the resource was used for a different amount of time than the wallDuration for the step. A compliant implementation MAY support this attribute if the element is supported.
- A consumable resource MAY have a consumptionRate attribute of type Float that indicates the average percentage that a resource was used over its wallDuration. For example, an overbooked SMP running 100 jobs across 32 processors might want to scale the usage and charge by the average fraction of processor usage actually delivered. A compliant implementation MAY support this attribute if the element is supported.

Simple consumable resources are listed in the table below:

Table M-8: Consumable Resource Node Properties

Element Name	Type	Description	Appearance	Compliance	Categories
Processors	Integer	Number of processors.	MAY	MUST	CAU
Memory	Float	Amount of memory.	MAY	SHOULD	CAU

Element Name	Type	Description	Appearance	Compliance	Categories
Disk	Float	Amount of disk.	MAY	SHOULD	CAU
Swap	Float	Amount of virtual memory.	MAY	MAY	CAU
Network	Float	Amount of network.	MAY	MAY	CAU

The following are two examples for specifying a consumable resource:

```
<Memory metric="Max" units="GB">483</Memory>
<Processors wallDuration="1234" consumptionRate="0.63">4</Processors>
```

Resource Element

In addition to the consumable resources enumerated in the above table, an extensible consumable resource is defined by the Resource element:

- A compliant implementation SHOULD support this element.
- This element MAY appear zero or more times within a given set of node properties.
- Like the other consumable resources, this property MUST be categorized as a configured, available or utilized property by being encompassed in the appropriate elements.
- This element is of type Float.
- It shares the other same properties and attributes as the other consumable resources but it requires an additional name (and optional type) attribute to describe it.
- This element MUST have a name attribute of type String that indicates the type of consumable resource being measured. A compliant implementation MUST support this attribute if the element is supported.
- This element MAY have a type attribute of type String that distinguishes it within a general resource class. A compliant implementation SHOULD support this attribute if the element is supported.

The following are two examples for specifying a `Resource` element:

```
<Resource name="License" type="MATLAB">1</Resource>
<Resource name="Telescope" type="Zoom2000" wallDuration="750"
metric="KX">10</Resource>
```

4.4 Node Reference

When a simple reference to a predefined node is needed in an encapsulating element, a Node element is used with the text content being the node ID:

```
<Node>node1</Node>
```

- This element MAY have an aggregation attribute of type String that provides a way to indicate multiple values with a single expression. A compliant implementation MAY support the aggregation attribute if the Feature element is supported. Possible values for this attribute include:
    - List a comma-separated list of features.
    - Pattern a regular expression (perl5) matching desired features.
    - Range a range of nodes of the form: `<prefix>[5-23,77]`.
- If an aggregation attribute is specified with the value of List, this element MAY also have a delimiter attribute of type String that indicates what delimiter is used to separate list elements. The default list delimiter is a comma.
- This element MAY have a count attribute of type Integer that indicates the instance count of the specified node(s).

The following is another example of a `Node` element:

```
<Node aggregation=”Pattern”>node[1-5]</Node>
```

5.0 Units of Measure Abbreviations

Abbreviation	Definition	Quantity
B	byte	1 byte
KB	Kilobyte	2^10 bytes
MB	Megabyte	2^20 bytes
GB	Gigabyte	2^30 bytes
TB	Terabyte	2^40 bytes
PB	Petabyte	2^50 bytes
EB	Exabyte	2^60 bytes

Abbreviation	Definition	Quantity
ZB	Aettabyte	2^70 bytes
YB	Yottabyte	2^80 bytes
NB	Nonabyte	2^90 bytes
DB	Doggabyte	2^100 bytes

M.4 Scalable Systems Software Resource Management and Accounting Protocol (SSSRMAP) Wire Protocol

Resource Management Interface Specs
Release Version 3.0.3
13 May 2004

Scott Jackson
Brett Bode
David Jackson
Kevin Walker

Status of this Memo

This is a specification defining a wire level protocol used between Scalable Systems Software components. It is intended that this specification will continue to evolve as these interfaces are implemented and thoroughly tested by time and experience.

Abstract

This document is a specification describing a connection-oriented XML-based application layer client-server protocol for the interaction of resource management and accounting software components developed as part of the Scalable Systems Software Center. The SSSRMAP Wire Protocol defines a framing protocol that includes provisions for security. The protocol is specified in XML Schema Definition and rides on the HTTP protocol.

Table of Contents

1.0 Introduction

A major objective of the Scalable Systems Software [SSS] Center is to create a scalable and modular infrastructure for resource management and accounting on terascale clusters including resource scheduling, grid-scheduling, node daemon support, comprehensive usage accounting and user interfaces emphasizing portability to terascale vendor operating systems. Existing resource management and accounting components feature disparate APIs (Application Programming Interfaces) requiring various forms of application coding to interact with other components.

This document proposes a wire level protocol expressed in an XML envelope to be considered as the foundation of a standard for communications between and among resource management and accounting software components. Individual components additionally need to define the particular XML binding necessary to represent the message format for communicating with the component.

2.0 Conventions Used in this Document

2.1 Keywords

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC2119.

2.2 XML Case Conventions

In order to enforce a consistent capitalization and naming convention across all SSSRMAP specifications 'Upper Camel Case' (UCC) and 'Lower Camel Case' (LCC) Capitalization styles shall be used. UCC style capitalizes the first character of each word and compounds the name. LCC style capitalizes the first character of each word except the first word. [XML_CONV][FED_XML]

1. SSSRMAP XML Schema and XML instance documents SHALL use the following conventions:
   - Element names SHALL be in UCC convention (example: `<UpperCamelCaseElement/>`.
   - Attribute names SHALL be in LCC convention (example: `<UpperCamelCaseElement lowerCamelCaseAttribute=”Whatever”/>`.
2. General rules for all names are:
   - Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
   - Underscores (_), periods (.) and dashes (-) MUST NOT be used (example: use JobId instead of JOB.ID, Job_ID or job-id).

2.3 Schema Definitions

```
SSSRMAP Schema Definitions appear like this
```

In case of disagreement between the schema file and this specification, the schema file takes precedence.

3.0 Encoding

Encoding tells how a message is represented when exchanged. SSSRMAP data exchange messages SHALL be defined in terms of XML schema [XML_SCHEMA].

3.1 Schema Header and Namespaces

The header of the schema definition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
  xmlns="https://www.w3.org/201/XMLSchema"
  xmlns:sssrmap="https://www.scidac.org/ScalableSystems/SSSRMAP"
  targetNamespace="https://www.scidac.org/ScalableSystems/SSSRMAP"
  elementFormDefault="qualified">
```

3.2 The Envelope Element

SSSRMAP messages and replies are encapsulated in the `Envelope` element. There are two possibilities for the contents of this element. If the contents are unencrypted, this element MUST contain a `Body` element and MAY contain a `Signature` element (refer to the section on Security). If the contents are encrypted, this element MUST contain exactly one `EncryptedData` element (refer to the section on Security). The `Envelope` element MAY contain namespace and other xsd-specific information necessary to validate the document against the schema. In addition, it MAY have any of the following attributes, which might serve as processing clues to the parser:

Attribute	Description
type	A message type providing a hint as to the body contents such as 'Request' or 'Notification'.
component	A component type such as 'QueueManager' or 'LocalScheduler'.
name	A component name such as 'OpenPBS' or 'Maui'.
version	A component version such as '2.2p12' or '3.2.2'.

```
<complexType name=EnvelopeType">
  <choice minOccurs="1" maxOccurs="1">
    <choice minOccurs="1" maxOccurs="2">
      <element ref="sssrmap:Signature" minOccurs="0" maxOccurs="1"/>
      <element ref="sssrmap:Body" minOccurs="1" maxOccurs="1"/>
    </choice>
    <element ref="sssrmap:EncryptedData" minOccurs="1" maxOccurs="1"/>
  </choice>
  <attribute name="type" type="string" use="optional"/>
  <attribute name="component" type="string" use="optional"/>
  <attribute name="name" type="string" use="optional"/>
  <attribute name="version" type="string" use="optional"/>
</complexType>

<element name="Envelope" type="sssrmap:EnvelopeType"/>
```

3.3 The Body Element

SSSRMAP messages and replies are encapsulated in the `Body` element. This element MUST contain exactly one `Request` or `Response` element.

```
<complexType name="BodyType">
  <choice minOccurs="1" maxOccurs="1">
    <element ref="sssrmap:Request" minOccurs="0" maxOccurs="1"/>
    <element ref="sssrmap:Response" minOccurs="0" maxOccurs="1"/>
    <any minOccurs="0" maxOccurs="1" namespace="##other"/>
  </choice>
</complexType>

<element name="Body" type="sssrmap:BodyType"/>
```

4.0 Transport Layer

This protocol will be built over the connection-oriented reliable transport layer TCP/IP. Support for other transport layers could also be considered, but native support for TCP/IP can be found on most terascale clusters and automatically handles issues such as reliability and connection fullness for the application developer implementing the SSSRMAP protocol.

5.0 Framing

Framing specifies how the beginning and ending of each message is delimited. Given that the encoding will be expressed as one or more XML documents, clients and servers need to know when an XML document has been fully read in order to be parsed and acted upon.

SSSRMAP uses the HTTP 1.1 [HTTP] protocol for framing. HTTP uses a byte-counting mechanism to delimit the message segments. HTTP chunked encoding is used. This allows for optional support for batched messages, large message segmentation and persistent connections.

5.1 Message Header Requirements

The HTTP request line (first line of the HTTP request header) begins with POST and is followed by a URI and the version of the HTTP protocol that the client understands. It is suggested for this protocol that the URI consist of a single slash, followed by the protocol name in uppercase (i.e., /SSSRMAP), though this field is not checked and could be empty, a single slash or any URI.

The Content-Type must be specified as test/xml. Charset can be optionally specified and defaults to US-ASCII. We recommend that charset be specified as 'utf-8' for maximum interoperability.

The Transfer-Encoding must be specified as chunked. The Content-Length must NOT be specified as the chunk size is specified in the message chunk.

Other properties such as User-Agent, Host, and Date are strictly optional.

5.2 Message Chunk Format

A message chunk consists of a chunk size in hexadecimal format (whose value is the number of bytes in the XML message not including the chunk size and delimiter) delimited by a CR/LF “\r\n” and followed by the message payload in XML that consists of a single XML document having a root element of `Envelope`.

5.3 Reply Header Requirements

The HTTP response line (first line of the HTTP response header) begins with HTTP and a version number, followed by a numeric code and a message indicating what sort of response is made. These response codes and messages indicate the status of the entire response and are as defined by the HTTP standard. The most common response is 200 OK, indicating that the message was received and an appropriate response is being returned.

The Content-Type must be specified as text/xml. Charset can be optionally specified and defaults to US-ASCII. We recommend that charset be specified as 'utf-8' for maximum interoperability.

The Transfer-Encoding MUST be specified as chunked. The Content-Length must NOT be specified.

Other properties such as Server, Host, and Date are strictly optional.

5.4 Reply Chunk Format

A reply chunk consists of a chunk size in hexadecimal format (whose value is the number of bytes in the XML reply not including the chunk size and delimiter) delimited by a CR/LF “\r\n” and followed by the reply payload in XML that consists of a single XML document having a root element of `Envelope`.

5.5 Message and Reply Tail Requirements and Multiple Chunks

This specification only requires that single chunks be supported. A server can optionally be configured to handle requests with persistent connections (multiple chunks). It will be the responsibility of clients to know whether a particular server supports this additional functionality. After all chunks have been sent, a connection is terminated by sending a zero followed by a carriage return-linefeed combination (0\r\n) and closing the connection.

5.6 Examples

Sample SSSRMAP Message Embedded in HTTP Request

```
POST /SSSRMAP HTTP/1.1\r\n
Content-Type: text/xml; charset=”utf-8”\r\n
Transfer-Encoding: chunked\r\n
\r\n
9A\r\n
<Envelope …/>
0\r\n
```

Sample SSSRMAP Reply Embedded in HTTP Response

```
HTTP/1.1 200 OK\r\n
Content-Type: text/xml; charset=”utf-8”\r\n
Transfer-Encoding: chunked\r\n
\r\n
2B4\r\n
<Envelope …/>
0\r\n
```

6.0 Asynchrony

Asynchrony (or multiplexing) allows for the handling of independent exchanges over the same connection. A widely-implemented approach is to allow pipelining (or boxcarring) by aggregating requests or responses within the body of the message or via persistent connections and chunking in HTTP 1.1. Pipelining helps reduce network latency by allowing a client to make multiple requests of a server, but requires the requests to be processed serially [RFC3117]. Parallelism could be employed to further reduce server latency by allowing multiple requests to be processed in parallel by multi-threaded applications.

Segmentation can become necessary if the messages are larger than the available window. With support for segmentation, the octet-counting requirement that you need to know the length of the whole message before sending it can be relegated to the segment level – and you can start sending segments before the whole message is available. Segmentation is facilitated via 'chunking' in HTTP 1.1.

The current SSSRMAP strategy supports only a single request or response within the Body element. A server can optionally support persistent connections from a client via HTTP chunking. Segmentation of large responses is also optionally supported via HTTP chunking. Later versions of the protocol could allow pipelined requests and responses in a single Body element.

7.0 Security

SSSRMAP security features include capabilities for integrity, authentication, confidentiality, and non-repudiation. The absence or presence of the various security features depend upon the type of security token used and the protection methods you choose to specify in the request.

For compatibility reasons, SSSRMAP specifies six supported security token types. Extensibility features are included allowing an implementation to use alternative security algorithms and security tokens. It is also possible for an implementation to ignore security features if it is deemed nonessential for the component. However, it is highly RECOMMENDED that an implementation support at least the default security token type in both authentication and encryption.

7.1 Security Token

A security token can be included in either the Signature block, and/or in the EncryptedData block (both described later) as an implicit or explicit cryptographic key. If this element is omitted, the security token is assumed to be a secret key shared between the client and the server.

The SecurityToken Element

This element is of type String. If the security token conveys an explicit key, this element's content is the value of the key. If the key is natively expressed in a binary form, it must be converted to base64 encoding as defined in XML Digital Signatures. If the type is not specified, it is assumed to be of type 'Symmetric'.

It can have any of the following optional attributes:

Attribute	Description
type	The type of security token (described subsequently): • A `type` attribute of 'Symmetric' specifies a shared secret key between the client and server. This is the default. • A `type` attribute of 'Asymmetric' specifies the use of public private key pairs between the client and server. • A `type` attribute of 'Password' encrypts and authenticates with a user password known to both the client and server. • A `type` attribute of 'Cleartext' allows the passing of a cleartext username and password and depends on the use of a secure transport (such as SSL or IPSec). • A `type` attribute of 'Kerberos5' specifies a kerberos token. • A `type` attribute of 'X509v3' specifies an X.509 certificate.

Attribute	Description
name	The name of the security token that serves as an identifier for the actor making the request (useful when the key is a password, or when the key value is implicit as when a public key is named but not included).

```
<complexType name="SecurityTokenType" mixed="true">
  <simpleContent>
    <extension base="string">
      <attribute name="type" type="string" use="optional">
      <attribute name="name" type="string" use="optional">
    </extension>
  </simpleContent>
</complexType>

<element name="SecurityToken" type="sssrmap:SecurityTokenType"/>
```

Security Token Types

SSSRMAP defines six standard security token types:

Symmetric Key

The default security token specifies the use of a shared secret key. The secret key is up to 128-bits long and known by both client and server. When using a symmetric key as a security token, it is not necessary to specify the `type` attribute with value 'Symmetric' because this is assumed when the attribute is absent. The `name` attribute should be specified indicating the actor issuing the request. If the user provides a password to be sent to the server for authentication, then the password is encrypted with the secret key using a default `method=”kw-tripledes”` (XML ENCRYPTION), base64 encoded and included as the string content of the `SecurityToken` element. If the client authenticated the user, then the `SecurityToken` element is empty. The same symmetric key is used in both authentication and encryption.

Asymmetric Key

Public and private key pairs can be used to provide non-repudiation of the client (or server). The client and the server must each have their own asymmetric key pairs. This mode is indicated by specifying the type attribute as 'Asymmetric'. The `name` attribute should be specified indicating the actor issuing the request. If the user provides a password to be sent to the server for authentication, then the password is encrypted with the server’s public key using a default `method=”rsa-1_5”` (XML ENCRYPTION), base64 encoded and included as the string content of the *SecurityToken* element. If the client authenticated the user, then the `SecurityToken` element is empty .The sender’s private key is used in authentication (signing) while the recipient’s public key is used for encryption.

Password

This mode allows for a username password combination to be used under the assumption that the server also knows the password for the user. This security token type is indicated by specifying a value of 'Password' for the *type* attribute. The password itself is used as the cryptographic key for authentication and encryption. The *name* attribute contains the user name of the actor making the request. The *SecurityToken* element itself is empty.

Cleartext

This security mode is equivalent to passing the username and password in the clear and depends upon the use of a secure transport (such as SSL or IPSec). The purpose of including this security token type is to enable authentication to occur from browsers over SSL or over internal LANs who use IPSec to encrypt all traffic. The password (or a hash of the password like in /etc/passwd) would have to be known by the server for authentication to occur. In this mode, neither encryption nor signing of the hash is performed at the application layer. This mode is indicated by specifying a value of 'Cleartext' for the *type* attribute. The `name` attribute contains the user name of the actor making the request and the string content of the `SecurityToken` element is the unencrypted plaintext password.

Kerberos

The use of a Kerberos version 5 token is indicated by specifying 'Kerberos5' in the `type` attribute. The `name` attribute is used to contain the kerberos user ID of the actor making the request. The `SecurityToken` element contains two sub elements. The *Authenticator* element contains the authenticator encoded in base64. A `Ticket` element contains the service-granting ticket, also base64 encoded.

GSI (X.509)

The Grid Security Infrastructure (GSI), which is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol can be indicated by specifying a `type` attribute of 'X509v3'. The name attribute contains the userid used that the actor was mapped to in the local system. The string content of the `SecurityToken` element is the GSI authentication message including the X.509 identity of the sender encoded in base64.

Example

```
<SecurityToken type=”Asymmetric” name=”scottmo”>
MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
</SecurityToken>
```

7.2 Authentication

Authentication entails how the peers at each end of the connection are identified and verified. Authentication is optional in an SSSRMAP message or reply. SSSRMAP uses a digital signature scheme for authentication that borrows from concepts in XML Digital

Signatures [XML_DSIG]. In addition to authentication, the use of digital signatures also ensures integrity of the message, protecting exchanges from third-party modification.

When authentication is used, a `Signature` element is prepended as the first element within the `Envelope` element. All of the security modes will create a digest of the data for integrity checking and store this in base64 encoding in a `DigestValue` element as a child of the `Signature` element. The digital signature is created by encrypting the hash with the appropriate security token and storing this value in a `SignatureValue` element as a child of the `Signature` element. The security token itself is included as a child of the `Security` element within a `SecurityToken` element.

There are a number of procedural practices that must be followed in order to standardize this approach. The digest (or hash) is created over the contents of the `Envelope` element (not including the Element tag or its attributes). This might be over one or more `Request` or `Notify` elements (or `Response` or `Ack` elements) and necessarily excludes the `Signature` Element itself. (Note that any data encryption is performed after the creation of the digital signature and any decryption is performed before authenticating so the `EncryptedData` element will not interfere with this process. Therefore, the signature is always based on the (hashed but) unencrypted data). For the purposes of generating the digest over the same value, it is assumed that the data is first canonicalized to remove extraneous whitespace, comments, etc., according to the XML Digital Signature algorithm and a transform is applied to remove namespace information. As a rule, any binary values are always transformed into their base64 encoded values when represented in XML.

The Signature Element

The `Signature` element MUST contain a `DigestValue` element that is used for integrity checking. It MUST also contain a `SecurityToken` element that is used to indicate the security mode and token type, and to verify the signature. It MUST contain a SignatureValue element that contains the base64 encrypted value of the signature wrought on the hash UNLESS the security token type indicates Cleartext mode where a signature would be of no value with the encryption key being sent in the clear -- in this case we use the password itself for authentication.

```
<complexType name="SignatureType">
  <choice minOccurs="2" maxOccurs="3">
    <element ref="sssrmap:DigestValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:SignatureValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:SecurityToken" minOccurs="0" maxOccurs="1"/>
  </choice>
</complexType>

<element name="Signature" type="sssrmap:SignatureType"/>
```

The DigestValue Element

The `DigestValue` element contains the cryptographic digest of the message data. As described above, the hash is generated over the `Body` element. The data to be hashed must first be canonicalized and appropriately transformed before generating the digest since typically an application will read in the XML document into an internal binary form, then marshal (or serialize) the data into a string, which is passed as input to the hash algorithm. Different implementations marshal the data differently so it is necessary to convert this to a well-defined format before generating the digest or the clients will generate different digest values for the same XML. The SHA-1 [SHA-1] message digest algorithm SHALL be used as the default method for generating the digest. A *method* attribute is defined as an extensibility option in case an implementation wants to be able to specify alternative message digest algorithms.

It MAY have a method attribute:

Attribute	Description
method	The message digest algorithm: • A `method` attribute of 'sha1' specifies the SHA-1 message digest algorithm. This is the default and is implied if this attribute is omitted.

```
<complexType name="DigestValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="DigestValue" type="sssrmap:DigestValueType"/>
```

The SignatureValue Element

The SignatureValue element contains the digital signature that serves the authentication (and potentially non-repudiation) function. The string content of the SignatureValue element is a base64 encoding of the encrypted digest value. The HMAC algorithm [HMAC] based on the SHA1 message digest SHALL be used as the default message authentication code algorithm for user identification and message integrity. A `method` attribute is defined as an extensibility option in case an implementation wants to be able to specify alternative digital signature algorithms.

It MAY have a method attribute:

Attribute	Description
method	The digest signature algorithm: • A `method` attribute of 'hmac-sha1' specifies the HMAC SHA-1 digital signature algorithm. This is the default and is implied if this attribute is omitted.

```
<complexType name="SignatureValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="SignatureValue" type="sssrmap:SignatureValueType"/>
```

Signature Example

Pre-authentication:

```
<Envelope>
  <Body>
    <Request action=”Query” actor=”kenneth”>
      <Object>User</Object>
      <Get name=”EmailAddress”></Get>
      <Where name=”Name”>scott</Where>
    </Request>
  </Body>
</Envelope>
```

Post-authentication:

```
<Envelope>
  <Signature>
    <DigestValue>
      LyLsF0Pi4wPU...
    </DigestValue>
    <SignatureValue>
      DJbchm5gK...
    </SignatureValue>
    <SecurityToken type=”Asymmetric” name=”kenneth”>
      MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
    </SecurityToken>
  </Signature>
  <Body>
    <Request action=”Query” actor=”kenneth”>
      <Object>User</Object>
      <Get name=”EmailAddress”></Get>
      <Where name=”Name”>scottmo</Where>
    </Request>
  </Body>
</Envelope>
```

7.3 Confidentiality

Confidentiality involves encrypting the sensitive data in the message, protecting exchanges against third-party interception and modification. Confidentiality is optional in an SSSRMAP message or reply. When confidentiality is required, SSSRMAP sessions use block cipher encryption with concepts borrowed from the emerging XML Encryption [XML_ENC] standard.

When confidentiality is used, encryption is performed over all child elements of the `Envelope` element (i.e., on the message data and any signature). The encrypted data is not signed -- rather the signature is encrypted. This data is replaced in-place within the envelope with an `EncryptedData` element. The data is first compressed using the gzip algorithm [ZIP]. Instead of encrypting this compressed data with the security token directly, a 192-bit random session key is generated by the sender and used to perform symmetric encryption on the compressed data. This key is itself encrypted with the security token and included with the encrypted data as the value of the `EncryptedKey` element as a child of the `EncryptedData` element. The ciphertext resulting from the data being encrypted with the session key is passed as the value of a `CipherValue` element (also a child of the `EncryptedData` element). As in the case with authentication, the security token itself is included as a child of the `Security` element within a `SecurityToken` element.

The EncryptedData Element

When SSSRMAP confidentiality is required, the `EncryptedData` element MUST appear as the only child element in the Envelope element. It directly replaces the contents of these elements including the data and any digital signature. It MUST contain an `EncryptedKey` element that is used to encrypt the data. It MUST contain a `CipherValue` element that holds the base64 encoded ciphertext. It MAY also contain a `SecurityToken` element that is used to indicate the security mode and token type. If the `SecurityToken` element is omitted, a Symmetric key token type is assumed. Confidentiality is not used when a security token type of 'Cleartext' is specified since it is pointless to encrypt the data with the encryption key in the clear.

```
<complexType name="EncryptionDataType">
  <choice minOccurs="0" maxOccurs="1">
    <element ref="sssrmap:EncryptedKey" minOccurs="1" maxOccurs="1"/>
    <element ref="sssrmap:CipherValue" minOccurs="1" maxOccurs="1"/>
    <element ref="sssmap:SecurityToken" minOccurs="1" maxOccurs="1"/>
  </choice>
</complexType>

<element name="EncryptedData" type="sssrmap:EncryptedDataType"/>
```

The EncryptedKey Element

The `EncryptedKey` element is a random session key encrypted with the security token. This approach is used for a couple of reasons. In the case where public key encryption is used, asymmetric encryption is much slower than symmetric encryption and it makes sense to use a symmetric key for encryption and pass along it along by encrypting it with the recipient's public key. It is also useful in that the security token, which does not change very often (compared to the session key, which changes for every connection) is used on a very small sampling of data (the session key), whereas if it was used to encrypt the whole message an attacker could more effectively exploit an attack against the ciphertext. The CMS Triple DES Key Wrap algorithm 'kw-tripledes' SHALL be used as the default method for key encryption. The session key is encrypted using the security token, base64 encoded and specified as the string content of the `EncryptedKey` element. A `method` attribute is defined as an extensibility option in case an implementation wants to be able to specify alternative key encryption algorithms.

It is REQUIRED that an implementation use a cryptographically secure Pseudo-Random number generator. we recommend that the session key be cryptographically generated (such as cyclic encryption, DES OFB, ANSI X9.17 PRNG, SHA1PRNG, or ANSI X12.17 (used by PGP)).

It MAY have a method attribute:

Attribute	Description
method	The key encryption algorithm: • A `method` attribute of 'kw-tripledes' specifies the CMS Triple DES Key Wrap algorithm. This algorithm is specified by the XML Encryption [XML_ENC] URI. It involves two Triple DES encryptions, a random and known Initialization Vector (IV) and a CMS key checksum. A 192-bit key encryption key is generated from the security token, lengthened as necessary by zero-padding. No additional padding is performed in the encryptions. This is the default and is implied if this attribute is omitted.

```
<complexType name="EncryptedKeyType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="EncryptedKey" type="sssrmap:EncryptedKeyType"/>
```

The CipherValue Element

The CipherValue element contains the message (and possibly signature) data encrypted with the random session key. The ciphertext is compressed using the gzip algorithm [ZIP], encrypted by the designated method, base64 encoded and included as the string content of the CipherValue element. The Triple DES algorithm with Cipher Block Chaining (CBC) feedback mode SHALL be used as the default method for encryption. A *method* attribute is defined as an extensibility option in case an implementation wants to be able to specify alternative data encryption algorithms.

It MAY have a method attribute:

Attribute	Description
method	The data encryption algorithm: • A `method` attribute of 'tripledes-cbc' specifies the Triple DES algorithm with Cipher Block Chaining (CBC) feedback mode. This algorithm is specified by the XML Encryption [XML_ENC] URI identifier. It specifies the use of a 192-bit encryption key and a 64-bit Initialization Vector (IV). Of the key bits, the first 64 are used in the first DES operation, the second 64 bits in the middle DES

Attribute	Description
	operation, and the third 64 bits in the last DES operation. The plaintext is first padded to a multiple of the block size (8 octets) using the padding scheme described in [XMLENC] for Block Encryption Algorithms (Padding per PKCS #5 will suffice for this). The resulting cipher text is prefixed by the IV. This is the default and is implied if this attribute is omitted.

```
<complexType name="CipherValueType">
  <simpleContent>
    <extension base="string">
      <attribute name="method" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<element name="CipherValue" type="sssrmap:CipherValueType"/>
```

Encryption Example

In this example, a simple request is demonstrated without a digital signature for the sake of emphasizing the encryption plaintext replacement.

Pre-encryption:

```
<Envelope>
  <Body>
    <Response>
      <Status>true</Status>
      <Code>000</Code>
      <Count>1</Count>
      <Data>
        <User>
          <EmailAddress>Scott.Jackson@pnl.gov</EmailAddress>
        </User>
      </Data>
    </Response>
  </Body>
</Envelope>
```

Post-encryption:

```
<Envelope>
  <EncryptedData>
    <EncryptedKey>
      NAkE9iQofYhyOfiHZ29kkEFVJ30CAwEAAaMSM...
    </EncryptedKey>
    <CipherValue>
      mPCadVfOMx1NzDaKMHNgFkR9upTW4kgBxyPW...
    </CipherValue>
    <SecurityToken type=”Asymmetric” name=”kenneth”>
      MIIEZzCCA9CggAwIBAgIQEmtJZc0rqrKh5i...
    </SecurityToken>
  </EncryptedData>
</Envelope>
```

8.0 References

[RFC2119] S. Bradner, “Key Words for Use in RFCs to Indicate Requirement Levels”, RFC 2119, March 1997.

[BEEP] M. Rose, “The Blocks Extensible Exchange Protocol Core”, RFC 3080, March 2001.

[HMAC] H. Krawczyk, M. Bellare, R. Canetti, “HMAC, Keyed-Hashing for Message Authentication”, RFC 2104, February 1997.

[SHA-1] U.S. Department of Commerce/National Institute of Standards and Technology, “Secure Hash Standard”, FIPS PUB 180-1.

[SSS] “Scalable Systems Software”, https://www.scidac.org/ScalableSystems

[HTTP] “Hypertext Transfer Protocol – HTTP/1.1”, RFC 2616, June 1999.

[XML_CONV] “I-X and <I-N-CA> XML Conventions”.

[FED_XML] “U.S. Federal XML Guidelines”.

[RFC3117] M. Rose, “On the Design of Application Protocols”, Informational RFC 3117, November 2001.

[XML_DSIG] D. Eastlake, J. Reagle Jr., D. Solo, “XML Signature Syntax and Processing”, W3C Recommendation, 12 February 2002.

[XML_ENC] T. Imamura, B. Dillaway, E. Smon, “XML Encryption Syntax and Processing”, W3C Candidate Recommendation, 4 March 2002.

[XRP] E. Brunner-Williams, A. Damaraju, N. Zhang, “Extensible Registry Protocol (XRP)”, Internet Draft, expired August 2001.

[XML] Bray, T., et al, “Extensible Markup Language (XML) 1.0 (Second Edition)”, 6 October 2000.

[XML_SCHEMA] D. Beech, M. Maloney, N. Mendelshohn, “XML Schema Part 1: Structures Working Draft”, April 2000.

[ZIP] J. Gailly, M. Adler, “The gzip home page”, https://www.gzip.org/

Appendix N: Moab Resource Manager Language Interface

The Moab Resource Manager Language (formerly called WIKI) is the language that some resource managers use to communicate with Moab, specifically a native RM. Generally each line represents a single resource or workload in Moab. The line contains the name of the resource or workload followed by a set of `<attr>=<val>` pairs. Although the Moab RM Language follows the same data format for all resource managers, each resource manager type receives and returns it differently. For instructions and examples on using Moab RM Language with a native RM, see 11.5 Managing Resources Directly with the Native Resource Manager Interface respectively.

In this appendix:

N.1 Moab RM Language Socket Protocol Description

In this section:

N.1.1 Moab RM Language

Moab RM language is formerly known as WIKI. The Moab scheduler uses a simple protocol for socket connections to the user client and the resource manager as described below:

```
<SIZE><CHAR>
CK=<CKSUM><WS>TS=<TIMESTAMP><WS>AUTH=<AUTH><WS>DT=<DATA>
```

Attribute	Description
<SIZE>	8 character decimal ASCII representation of the size of the packet following '<SIZE><CHAR>' Leading zeroes must be used to pad this value to 8 characters if necessary.
<CHAR>	A single ASCII character.
<CKSUM>	A 16 character hexadecimal ASCII DES-based checksum calculated using the algorithm below* and <SEED> selected and kept secret by the site admins. The checksum is performed on the line from TS= to the end of the message including <DATA>.
<WS>	A series of white space characters consisting of either tabs and/or space characters.
<TIMESTAMP>	ASCII representation of epoch time.
<AUTH>	Identifier of user requesting service (i.e., USERNAME).
<DT>	Data to be sent.

An example header follows: `00001057 CK=cdf6d7a7ad45026f TS=922401962 AUTH=sched DT=<DATA>`

Where `<DATA>` is replaced by actual message data.

N.1.2 Checksum Algorithm ('C' Version)

Checksum Algorithm ('C' Version)

```
#define MAX_CKSUM_ITERATION 4

int GetChecksum(
 char *Buf,
 int BufSize,
 char *Checksum,
 char *CSKey) /* Note: pass in secret key */
 {
 unsigned int crc;
 unsigned int lword;
 unsigned int irword;
 int index;
 unsigned int Seed;
 Seed = (unsigned int)strtoul(CSKey,NULL,0);
 crc = 0;
 for (index = 0;index < BufSize;index++)
 {
 crc = (unsigned int)DoCRC((unsigned short)crc,Buf[index]);
 }
 lword = crc;
 irword = Seed;
 PSDES(&lword,&irword);
 sprintf(Checksum,"%08x%08x",
 lword,
 irword);
 return(SUCCESS);
 }


unsigned short DoCRC(
 unsigned short crc,
 unsigned char onech)
 {
 int index;
 unsigned int ans;
 ans = (crc ^ onech << 8);
 for (index = 0;index < 8;index++)
 {
 if (ans & 0x8000)
 ans = (ans <<= 1) ^ 4129;
 else
 ans <<= 1;
 }
 return((unsigned short)ans);
 }


int PSDES(
 unsigned int *lword,
 unsigned int *irword)
 {
 int index;
 unsigned int ia;
 unsigned int ib;
 unsigned int iswap;
 unsigned int itmph;
 unsigned int itmpl;
 static unsigned int c1[MAX_CKSUM_ITERATION] = {
 0xcba4e531, 0x537158eb, 0x145cdc3c, 0x0d3fdeb2 };
 static unsigned int c2[MAX_CKSUM_ITERATION] = {
 0x12be4590, 0xab54ce58, 0x6954c7a6, 0x15a2ca46 };
 itmph = 0;
 itmpl = 0;
 for (index = 0;index < MAX_CKSUM_ITERATION;index++)
 {
 iswap = *irword;
 ia = iswap ^ c1[index];
 itmpl = ia & 0xffff;
 itmph = ia >> 16;
 ib = (itmpl * itmpl) + ~(itmph*itmph);
 ia = (ib >> 16) | ((ib & 0xffff) << 16);
 *irword = (*lword) ^ ((ia ^ c2[index]) + (itmpl * itmph));
 *lword = iswap;
 }
 return(SUCCESS);
 }
```

N.1.3 Header Creation (PERL Code)

(taken from PNNL's QBank client code)

Header Creation (PERL Code)

```
################################################################################
#
# subroutine wiki($COMMAND)
#
# Sends command to Moab server and returns the parsed result and status
#
################################################################################
sub wiki
{
 my($COMMAND,$REQUEST,$result);
 my($sockaddr,$hostname);
 my($name,$aliases,$proto,$port,$type,$len,$thisaddr);
 my($thisport,$thatport,$response,$result);
 $COMMAND = shift;
 #
 # Establish socket connection
 #
 $sockaddr = 'S n a4 x8';
 chop ($hostname = `hostname`);
 ($name,$aliases,$proto)=getprotobyname('tcp');
 ($name,$aliases,$type,$len,$thisaddr)=gethostbyname($hostname);
 ($name,$aliases,$type,$len,$thataddr)=gethostbyname($BANKHOST);
 $thisport=pack($sockaddr, &AF_INET,0,$thisaddr);
 $thatport=pack($sockaddr, &AF_INET,$BANKPORT,$thataddr);
 socket(S, &PF_INET,&SOCK_STREAM,$proto) || die "cannot create socket\n";
 bind(S,$thisport) || die "cannot bind socket\n";
 connect(S,$thatport) || die "cannot connect socket\n";
 select(S); $| = 1; # Turn on autoflushing
 select(stdout); $| = 1; # Select STDOUT as default output
 #
 # Build and send command
 #
 $REQUEST="COMMAND=$COMMAND AUTH=$AUTH";
 chomp($CHECKSUM = `$QSUM "$REQUEST"`);
 $REQUEST .= " CHECKSUM=$CHECKSUM";
 my $command=pack "a8 a1 A*",sprintf("%08d",length($REQUEST))," ",$REQUEST;
 print S "$command"; # Send Command to server
 @REPLY=();
 while (~~) { push(@REPLY,$_); } # Listen for Reply~~
~~ $STATUS=grep(/STATUSCODE=(\d*)/&&$1,@REPLY); # STATUSCODE stored in $STATUS~~
~~ grep(s/.*RESULT=//,@REPLY); # Parse out the RESULT~~
~~ return @REPLY;~~
~~}~~
```

N.1.4 Header Processing (PERL Code)

Header Processing (PERL Code)

```
sysread(NS,$length,8); # Read length string
sysread(NS,$delimiter,1); # Read delimiter byte
$DEBUG && print STDERR "length=[$length]\tdelimiter=[$delimiter]\n";
while($length) {
 $DEBUG && print STDERR "Awaiting $length bytes -- ".`date`;
 $length-=sysread(NS,$request,$length); # Read request
 sleep 1;
 }
%REQUEST=();
chomp($request);
foreach (@REQUEST=&shellwords($request)) # Parse arguments into array
 {
 ($key,$value)=split(/=/,$_);
 $REQUEST{$key}=$value unless defined $REQUEST{$key};
 }
$request =~ s/\s+CHECKSUM=.*//; # Strip off the checksum
print STDERR "REQUEST=$request\n";
chomp($checksum=`$QSUM "$request"`);
$me=$REQUEST{AUTH};
$command=$REQUEST{COMMAND};
if (!grep($command eq $_,@VALIDCMDS))
 { $REPLY = "STATUSCODE=0 RESULT=$command is not a valid command\n";}
elsif ($checksum ne $REQUEST{CHECKSUM})
 { $REPLY = "STATUSCODE=0 RESULT=Invalid Checksum\n";}
else
 { $REPLY = do $command(@REQUEST); }
$len=sprintf("%08d",length($REPLY)-1);
$delim=' ';
$DEBUG && print STDERR "REPLY=${len}${delim}$REPLY\n";
$buf="$len"."$delim"."$REPLY";
syswrite(NS,$buf,length($buf));
close NS;
```

N.2 Moab Resource Manager Language Data Format

In this section:

N.2.1 Query Resources Data Format

Name	Format	Default	Description
ADISK	`<INTEGER>`	0	Available local disk on node (in MB).
AFS	`<fs id="X" size="X" io="Y" rcount="X" wcount="X" ocount="X"></fs>[...]`	0	Available filesystem state.

Name	Format	Default	Description
AMEMORY	`<INTEGER>`	0	Available/free RAM on node (in MB).
APROC	`<INTEGER>`	1	Available processors on node.
ARCH	`<STRING>`	---	Compute architecture of node.
ARES	One or more comma-delimited `<NAME>:<VALUE>` pairs (i.e., `MATLAB:6,COMPILER:100`)	---	Arbitrary consumable resources currently available on the node.
ASWAP	`<INTEGER>`	0	Available swap on node (in MB).
CCLASS	One or more bracket enclosed `<NAME>:<COUNT>` pairs (i.e., `[batch:5][sge:3]`)	---	Run classes supported by node. Typically, one class is 'consumed' per task. Therefore, an 8 processor node may have 8 instances of each class it supports present (i.e., `[batch:8][interactive:8]`).
CDISK	`<INTEGER>`	0	Configured local disk on node (in MB).
CFS	`<STRING>`	0	Configured filesystem state.
CMEMORY	`<INTEGER>`	0	Configured RAM on node (in MB).
CONTAINERNODE	`<STRING>`	---	The physical machine that is hosting the virtual machine. Only valid on VMs.
CPROC	`<INTEGER>`	1	Configured processors on node.
CPULOAD	`<DOUBLE>`	0.0	One minute BSD load

Name	Format	Default	Description
			average.
CPUSPEED	`<INTEGER>`	---	The node's processor speed (in MHz).
CRES	One or more comma-delimited `<NAME>:<VALUE>` pairs (i.e., `MATLAB:6,COMPILER:100`)	---	Arbitrary consumable resources supported and tracked on the node (i.e., software licenses or tape drives
CSWAP	`<INTEGER>`	0	Configured swap on node (in MB).
FEATURE	One or more colon-delimited `<STRING>`'s (i.e., `WIDE:HSM`) Punctuation and escapes are not allowed (.,:;\t\n\, etc.).	---	Generic attributes, often describing hardware or software features, associated with the node.
GEVENT	`GEVENT [ <EVENTNAME>]=<STRING>`	---	Generic event occurrence and context data.
GMETRIC	`GMETRIC [ <METRICNAME> ]=<DOUBLE>`	---	Current value of generic metric (i.e., '`GMETRIC [temp]=103.5`').
IDLETIME	`<INTEGER>`	---	Number of seconds since last detected keyboard or mouse activity (often used with desktop harvesting).
MAXTASK	`<INTEGER>`	`<CPROC>`	Maximum number of tasks allowed on the node at any given time.
NETADDR	`<STRING>`	---	The IP address of the machine.
NODEINDEX	`<INTEGER>`	---	The node's index.

Name	Format	Default	Description
OS	`<STRING>`	---	Operating system running on node.
OSLIST	One or more comma-delimited `<STRING>`'s with quotes if the string has spaces (i.e., `"SAS7 AS3 Core Baseline Build v0.1.0","RedHat AS3-U5Development Build v0.2"`).	---	Operating systems accepted by node.
OTHER	`<ATTR>=<VALUE> [,<ATTR>=<VALUE>]...`	---	Opaque node attributes assigned to node.
PARTITION	`<STRING>`	DEFAULT	Partition to which node belongs.
POWER	`<BOOLEAN>`		Whether the machine is on or off.
PRIORITY	`<INTEGER>`	---	Node allocation priority.
RACK	`<INTEGER>`	0	Rack location of the node.
SLOT	`<INTEGER>`	0	Slot location of the node.
STATE*	One of the following: `Idle`, `Running`, `Busy`, `Unknown`, `Drained`, `Draining`, or `Down`	Down	State of the node.
UPDATETIME*	`<EPOCHTIME>`	0	Time node information was last updated.
VARATTR	`<ATTR1>=<VAL1> [=<displayName1>] [+<ATTR2>=<VAL2> [=<displayName2>]]...`	---	Plus-delimited (+) list of `<ATTR>=<VAL> [=<displayName>]` pairs that jobs can request. You can replace any of the equals signs with colons if desired. Specifying a display name

Name	Format	Default	Description
			enables you to choose a name that will be displayed in the Mongo database instead of the unique ID (the <VALUE>). If you give two different attributes the same value and one of them also has a display name specified, both attributes will appear with the same display name.
VARIABLE	`<ATTR>=<VAL>`	---	Generic variables to be associated with node.
VMOSLIST	`<STRING>`	---	Comma-delimited list (`,`) of supported virtual machine operating systems for this node.
XRES	One or more comma-delimited `<NAME>:<VALUE>` pairs (i.e., `MATLAB:6,COMPILER:100`)	---	Amount of external usage of a particular generic resource.

* indicates required field

Node states have the following definitions:

State	Description
Busy	Node is running some jobs and will not accept additional jobs.
Down	Resource Manager problems have been detected. Node is incapable of running jobs.
Draining	Node is responding but will not accept new jobs.
Idle	Node is ready to run jobs but currently is not running any.

State	Description
Running	Node is running some jobs and will accept additional jobs.
Unknown	Node is capable of running jobs but the scheduler will need to determine if the node state is actually Idle, Running, or Busy.

N.2.2 Query Workload Data Format

Name	Format	Default	Description
ACCOUNT	`<STRING>`	---	AccountID associated with job.
ARGS	`<STRING>`	---	Job command-line arguments.
COMMENT	`<STRING>`	0	Job resource manager extension arguments including qos, dependencies, reservation constraints, etc.
COMPLETETIME*	`<EPOCHTIME>`	0	Time job completed execution.
DDISK	`<INTEGER>`	0	Quantity of local disk space (in MB) that must be dedicated to each task of the job.
DGRES	`name:value [,name:value]`	---	Dedicated generic resources per task.
DPROCS	`<INTEGER>`	1	Number of processors dedicated to the job.
DSWAP	`<INTEGER>`	0	Quantity of virtual memory (swap, in MB) that must be dedicated to each task of the job.
ENDDATE	`<EPOCHTIME>`	[ANY]	Time by which job must complete.
ENV	`<STRING>`	---	Job environment variables.
ERROR	`<STRING>`	---	File to contain STDERR.
EVENT	`<EVENT>`	---	Event or exception experienced by job.

Name	Format	Default	Description
EXEC	`<STRING>`	---	Job executable command.
EXITCODE	`<INTEGER>`	---	Job exit code.
FLAGS	`<STRING>`	---	Job flags.
GEOMETRY	`<STRING>`	---	String describing task geometry required by job.
GNAME*	`<STRING>`	---	GroupID under which job will run.
HOSTLIST	Comma- or colon-delimited list of hostnames - suffix the hostlist with a carat (^) to mean superset; suffix with an asterisk (*) to mean subset; otherwise, the hostlist is interpreted as an exact set	`[ANY]`	List of required hosts on which job must run (see TASKLIST below). A subset means the specified hostlist is used first to select hosts for the job. If the job requires more hosts than are in the hostlist, they will be obtained from elsewhere if possible. If the job does not require all of the jobs in the hostlist, it will use only the ones it needs. A superset means the hostlist is the *only* source of hosts that should be considered for running the job. If the job can't find the necessary resources in the hosts in this list it should *not* run. No other hosts should be considered in allocating the job.
INPUT	`<STRING>`	---	File containing STDIN.
IWD	`<STRING>`	---	Job's initial working directory.
NAME	`<STRING>`	---	User specified name of job.
NODES	`<INTEGER>`	1	Number of nodes required by job (see 2.3.2 Nodes for more information).
OUTPUT	`<STRING>`	---	File to contain STDOUT.
PARTITIONMASK	One or more colon-delimited `<STRING>`s	`[ANY]`	List of partitions where job can run.

Name	Format	Default	Description
PREF	Colon-delimited list of `<STRING>`s	---	List of preferred node features or variables. See the job extension PREF for more information.
PRIORITY	`<INTEGER>`	---	System priority (absolute or relative - use '+' and '–' to specify relative).
QOS	`<INTEGER>`	0	Quality of service requested.
QUEUETIME*	`<EPOCHTIME>`	0	Time job was submitted to resource manager.
RARCH	`<STRING>`	---	Architecture required by job.
RCLASS	List of bracket enclosed `<STRING>` `:<INTEGER>` pairs	---	List of `<CLASSNAME>:<COUNT>` pairs indicating type and number of class instances required per task (i.e., `[batch:1]` or `[batch:2]` `[tape:1]`).
RDISK	`<INTEGER>`	0	Local disk space (in MB) required to be configured on nodes allocated to the job.
RDISKCMP	One of >=, >, ==, <, or <=	>=	Local disk comparison (i.e., node must have > 2048 MB local disk).
REJCODE	`<INTEGER>`	0	Reason job was rejected.
REJCOUNT	`<INTEGER>`	0	Number of times job was rejected.
REJMESSAGE	`<STRING>`	---	Text description of reason job was rejected.
REQRSV	<STRING>	---	Name of reservation where job must run.
RESACCESS	<STRING>	---	List of reservations where job can run.
RFEATURES	Colon-delimited list <STRING>'s	---	List of features required on nodes.

Name	Format	Default	Description
RMEM	<INTEGER>	0	Real memory (RAM, in MB) required to be configured on nodes allocated to the job.
RMEMCMP	One of '>=', '>', '==', '<', or '<='	>=	Real memory comparison (i.e., node must have >= 512MB RAM).
ROPSYS	<STRING>	---	Operating system required by job.
RSOFTWARE	<RESTYPE>[{+|:}<COUNT>][@<TIMEFRAME>]	---	Software required by job.
RSWAP	<INTEGER>	0	Virtual memory (swap, in MB) required to be configured on nodes allocated to the job.
RSWAPCMP	One of '>=', '>', '==', '<', or '<='	>=	Virtual memory comparison (i.e., node must have ==4096 MB virtual memory).
SID	<STRING>	---	System ID (global job system owner).
STARTDATE	<EPOCHTIME>	0	Earliest time job should be allowed to start.
STARTTIME*	<EPOCHTIME>	0	Time job was started by the resource manager.
STATE*	One of the following: `Idle`, `Running`, `Hold`, `Suspended`, `Completed`, or `Removed`	Idle	State of job.
SUSPENDTIME	<INTEGER>	0	Number of seconds job has been suspended.
TASKLIST	One or more comma-delimited <STRING>'s	---	List of **allocated** tasks, or in other words, comma-delimited list of node ID's associated with each active task of job (i.e., cl01, cl02, cl01, cl02, cl03). The tasklist is initially selected by the

Name	Format	Default	Description
			scheduler at the time the StartJob command is issued. The resource manager is then responsible for starting the job on these nodes and maintaining this task distribution information throughout the life of the job (see HOSTLIST above).
TASKS*	<INTEGER>	1	Number of tasks required by job (see the component Task Definition for more information).
TASKPERNODE	<INTEGER>	0	Exact number of tasks required per node.
UNAME*	<STRING>	---	UserID under which job will run.
UPDATETIME*	<EPOCHTIME>	0	Time job was last updated.
WCLIMIT*	[[HH:]MM:]SS	864000	Walltime required by job.

* indicates required field

Job states have the following definitions:

State	Definition
Completed	Job has completed.
Hold	Job is in the queue but is not allowed to run.
Idle	Job is ready to run.
Removed	Job has been canceled or otherwise terminated externally.
Running	Job is currently executing.
Suspended	Job has started but execution has temporarily been suspended.

Completed and canceled jobs should be maintained by the resource manager for a brief time, perhaps 1 to 5 minutes, before being purged. This provides the scheduler time to obtain all final job state information for scheduler statistics.

Related Topics

- 11.5 Managing Resources Directly with the Native Resource Manager Interface

Appendix O: SCHEDCFG Flags

Flag	Description
AGGREGATENODEFEATURES	Causes Moab to aggregate features reported by the different resource managers. For example, if you have two resource managers reporting different features for the same node, Moab will add both features together (instead of one being overwritten by the other). To set features manually, you can use *`mnodectl -m features`* (for details, see the command mnodectl).
ALLOWCREDENTIALSWITHSPACES	Lets Moab ignore POSIX standards and allows groups, users, and accounts with spaces in their names.
ALLOWINFINITEJOBS	Allows infinite wallclock times to be accepted. Previously, jobs with infinite job times were allowed by default.
ALLOWMULTICOMPUTE	Tells Moab how to resolve conflicting information from different resource managers. If specified, Moab will use the STATE and OS information from the resource manager that reports the node as online.
ALLOWPERJOBNODESETISOPTIONAL	Specifies whether Moab will read the NODESETISOPTIONAL resource manager extension on an individual job or use the global setting.
CANCELFAILEDDEPENDENCYJOBS	Automatically cancels dependency jobs that will never run because of an unmet requirement. For example, if you ran a job with both an afterok and afternotok job attached to it and that job was successful, the afterok job would run, leaving the afternotok job idle in the queue. If set, Moab will cancel the job with the failed dependency and remove it from the queue. For more information about job dependencies, see 9.5 Job Dependencies.

Flag	Description
	If you want to cancel all jobs that a specified `<job_id>` depends on, use `mjobctl -c flags=follow-dependency <job_id>` instead.
CHECKCIRCULARDEPENDENCIES	If you regularly submit job dependencies based on job names (and not job IDs) it is possible to accidentally create a circular dependency where jobs end up blocked. Moab can recursively check for circular dependencies when jobs are submitted by enabling this flag. Note that this check can be extensive depending on the workload submitted.
DISABLEPERJOBNODESETS	Disables a job's ability to override the system specified node set. See 13.3 Resource Manager Extensions for more information.
DISABLEPARTIALNODERESERVATIONS	Blocks partial node reservations.
ENABLEDYNAMICNODES	Enables the ability to automatically remove nodes from Moab that are no longer reported by the resource manager.
ENABLEJOBTRIGGERSONRSV	Enable job start triggers based on the job's reservation and not the actual start of the job (allows for negative offset job start triggers).
ENABLEMOABJOBENV	Puts the Moab job variables on every job.
ENFORCERESERVEDNODES	Without this flag Moab tries to optimize the reservation for a job before it starts, meaning a job can start on nodes that weren't part of its reservation. With this flag Moab tries to start jobs only on the nodes that were reserved.
ENFORCESAMENODESET	The same nodeset is not enforced across job requirements by default, rather each requirement is scheduled separately and the nodesets are determined on a per-req basis. To have Moab enforce the same nodeset across all job requirements set this flag.
EXTENDEDGROUPSUPPORT	Allows Moab to consider a user's secondary Linux

Flag	Description
	groups when dealing with reservation ACLs.
FASTGROUPLOOKUP	Moab will use the system call `getgrouplist` to gather group information. This can significantly improve performance on some LDAP systems.
FASTRSVSTARTUP	Speeds up start time if there are existing reservations. When you set the FASTRSVSTARTUP flag, Moab will also set the DISABLEPARTIALNODERESERVATIONS flag. FASTRSVSTARTUP is incompatible with partial node reservations. FASTRSVSTARTUP maintains the resource manager-reported node order at all times. On very large systems, if there is a reservation in the checkpoint file on all the nodes, it would take a really long time for Moab to start up. For every node in the reservation, Moab checks every other node. With this flag, Moab just uses the nodelist that was checkpointed to create the reservation. It speeds up the startup process because it doesn't have to check every node. Where Moab would take 8 - 10 minutes to start up with an 18,000 node reservation without the flag, Moab can start up in 2-3 minutes with the flag. With the flag you will see one difference in checknode. A reservation that uses all the procs on a node initially shows that all the procs are blocked. Without the flag, and as jobs fill on the node, the blocked resources will be configured - dedicated (ex. 5/6). With the flag, the blocked resources will always be what the reservation is blocking and won't change when jobs fill on the node. **Without Flag** Reservations: brian.1x1 User -00:12:52 -> INFINITY (INFINITY) Blocked Resources@-00:00:02 Procs: 5/6 (83.33%) Mem: 0/5000 (0.00%) Blocked Resources@00:04:58 Procs: 6/6

Flag	Description
	(100.00%) Mem: 0/5000 (0.00%) m.2x1 Job:Running -00:00:02 -> 00:04:58 (00:05:00) Jobs: m.2 **With Flag** Reservations: brian.1x1 User -00:00:15 -> INFINITY (INFINITY) Blocked Resources@-00:00:02 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%) Blocked Resources@00:04:58 Procs: 6/6 (100.00%) Mem: 0/5000 (0.00%) m.1x1 Job:Running -00:00:02 -> 00:04:58 (00:05:00) Jobs: m.1
FILELOCKHA	This is a High Availability feature. This prevents scheduling conflicts between multiple Moab servers.
FREECOMPLETEDJOBSUBMITSTRING	Moab frees the job submit string for completed jobs, decreasing the amount of memory needed during operation. This is useful in environments with large job scripts that can create a large memory footprint.
IGNOREPIDFILELOCK	Moab frees the job submit string and environment for completed jobs, decreasing the amount of memory needed during operation. This is useful in environments with large job scripts and environments that can create a large memory footprint.
INTERACTIVEWCACCURACY	Moab will assume all interactive jobs are 100% accurate with respect to walltime. This is useful for sites that don't enforce walltimes for interactive jobs but don't want users punished for inaccurate interactive job walltimes.
JOBSUSERSVWALLTIME	Allows jobs submitted without a walltime request or default walltime received from a class or queue but with an `ADVRES:reservation` to inherit their walltime limit from the reservation instead of the Moab default. The job walltime limit is then the remaining time of the reservation to which the job was submitted.

Flag	Description
NOCLASSUPDATE	While running against Torque, Moab will not update classes when it refreshes each iteration. Moab loads the classes at startup, but does not refresh them until the next time it is restarted.
NORMALIZETASKDEFINITIONS	Instructs Moab to normalize all tasks that it receives via an mshow -a command. Moab normalizes the task definition to one processor and then changes the tasks requested to the number of processors requested. For example, when the following is received by Moab: `mshow -a -w mintasks=1@procs:4+mem:4096` It is changed to this: `mshow -a -w mintasks=4@procs:1+,mem:1024,tpn=4`
OPTIMIZEDBACKFILL	Turns on an optimization within the `FIRSTFIT` backfill algorithm that checks whether there is a system-wide reservation blocking most jobs from running. This flag speeds up backfill scheduling (in the case where there is a system-wide reservation blocking most users) by checking access to the reservation sooner rather than later. This flag will be the default in future versions of Moab.
PRIORITYPOLICYBLOCKING	By default, a job that violates a policy is placed into the blocked queue. Jobs with a lower priority, but that do not violate the policy, will run. This can lead to situations where small jobs starve out larger, higher priority jobs. Moab allows the job that violates the policy to continue consuming the policy slots while it remains blocked. With the policy slots consumed, the smaller, lower priority jobs will not run. The higher priority job will continue to consume the policy slots until it has consumed enough to actually run. Note that because the blocked job consumes policy slots, this will inevitably lead to lower system utilization.
SETDEFAULTHOSTLISTEXACTSET	By default, all Hostlist requests will be interpreted as an exactset request. See the job extension HOSTLIST for more information.

Flag	Description
SHOWCOMPLETEDDEPENDENCIES	Continues showing dependencies on a job even after the dependencies have been satisfied.
SHOWREQUESTEDPROCS	Shows requested processors regardless of NodeAccessPolicy in showq. When `SINGLEJOB NODEACCESSPOLICY` is used and the job requests one processor, *`showq`* displays the job with one processor.
SHOWUSERJOBSONLY	Causes Moab, when a non-admin user runs *`showq`*, to return only that user's jobs. If an admin runs *`showq`* when this flag is set, Moab returns the jobs of all users; no restrictions are placed on admins.
STRICTSPOOLDIRPERMISSIONS	Enforces at least a 511 permission on the Moab spool directory.
SUSPENDEDJOBNODEBFINELIGIBLE	Turns off back fill on nodes where a job was suspended due to preemption. When enabled, CHECKSUSPENDEDJOBPRIORITY must be set to True. See the parameter CHECKSUSPENDEDJOBPRIORITY.
UNMIGRATEONDEFER	Forces Moab to unmigrate a job in a grid if it enters a deferred state.
USERMCOMPLETEDJOBSTATS	Calculate job statistics using the resource manager reported metrics rather than Moab's internal metrics.