Maui Administrator's Guide

Maui 3.2 Last Updated May 16

Overview

The Maui Scheduler can be thought of as a *policy engine* which allows sites control over when, where, and how resources such as processors, memory, and disk are allocated to jobs. In addition to this control, it also provides mechanisms which help to intelligently optimize the use of these resources, monitor system performance, help diagnose problems, and generally manage the system.

Table of Contents:

1.0 Philosophy

The goal of a scheduler in the broadest sense is to make users, administrators, and managers happy. Users desire the ability to specify resources, obtain quick turnaround on their jobs, and receive reliable allocation of resources. Administrators desire happy managers and happy users. They also desire the ability to understand both the workload and the resources available. This includes current state, problems, and statistics as well as information about what is happening *under the covers*. They need an extensive set of *buttons* and *knobs* to both enable management enforced policies and tune the system to obtain desired statistics.

1.1 Value of a Batch System

Batch systems provide a mechanism for submitting, launching, and tracking jobs on a shared resource. These services fullfil one of the major responsibilities of a batch system, providing centralized access to distributed resources. This greatly simplifies the use of the cluster's distributed resources allowing users a 'single system image' in terms of the management of their jobs and the aggregate compute resources available. However, batch systems must do much more than provide a global view of the cluster. As with many shared systems, complexities arise when attempting to utilize compute resources in a fair and effective manner. These complexities can lead to poor performance and significant inequalities in usage. With a batch system, a scheduler is assigned the job of determining, when, where, and how jobs are run so as to maximize the output of the cluster. These decisions are broken into three primary areas.

1.1.1 Traffic Control

A scheduler is responsible for preventing jobs from interfering with each other. If jobs are allowed to contend for resources, they will generally decrease the performance of the cluster, delay the execution of these jobs, and possibly cause one or more of the jobs to fail. The scheduler is responsible for internally tracking and dedicating requested resources to a job, thus preventing use of these resources by other jobs.

1.1.2 Mission Policies

When clusters or other HPC platforms are created, they are typically created for one or more specific purposes. These purposes, or mission goals, often define various rules about how the system should be used and who or what will be allowed to use it. To be effective, a scheduler must provide a suite of policies which allow a site to *map* site mission policies into scheduling behavior.

1.1.3 Optimizations

The compute power of a cluster is a limited resource and over time, demand will inevitably exceed supply. Intelligent scheduling decisions can significantly improve the effectiveness of the cluster resulting in more jobs being run and quicker job turnaround. Subject to the constraints of the traffic control and mission policies, it is the job of the scheduler to use

whatever freedom is available to schedule jobs in such a manner so as to maximize cluster performance.

1.2 Philosophy and Goals of the Maui Scheduler

Managers desire maximum return on investment often meaning high system utilization and the ability to deliver various qualities of service to various users and groups. They need to understand how the available resources are being delivered to the various users over time and need the ability to have the administrators tune 'cycle delivery' to satisfy the current site mission objectives.

How well a scheduler succeeds can only be determined if various metrics are established and a means to measure these metrics are available. While statistics are important, their value is limited unless optimal statistical values are also known for the current environment including workload, resources, and policies. If one could determine that a site's typical workload obtained an average queue time of 3 hours on a particular system, this would be a good *statistic*. However, if one knew that through proper tuning, the system could deliver an average queue time of 1.2 hours with minimal negative side effects, this would be valuable *knowledge*.

The Maui Scheduler was developed with extensive feedback from users, administrators, and managers. At its core, it is a tool designed to truly *manage* resources and provide meaningful information about what is actually happening on the system. It was created to satisfy real-world needs of a batch system administrator as he tries to balance the needs of users, staff, and managers while trying to maintain his sanity.

2.0 Installation

Maui installation consists of the following steps:

- 2.1 Maui Installation
- 2.2 Initial Maui Configuration
- 2.3 Testing

2.1 Maui Installation

Building Maui

To install Maui, untar the distribution file, enter the **maui-<VERSION>** directory, then run **configure** and **make** as shown in the example below:

```
> gtar -xzvf maui-3.0.7.tar.gz
> cd maui-3.0.7
> ./configure
> make
```

Installing Maui (Optional)

When you are ready to use Maui in production, you may install it into the install directory you have configured using **make install**

```
> make install
```

Note: Until the *install* step is performed, all Maui executables will be placed in **$MAUIHOMEDIR/bin**. (i.e., maui-3.0.7/bin in the above example)

2.2 Initial Maui Configuration

After you install Maui, there are a few decisions which must be made and some corresponding information which will need to be provided in the Maui configuration file, **maui.cfg**. The **configure** script automatically sets most of these parameters for you. However, this document provides some additional information to allow further initial configuration. If you are satisfied with the values specified in **configure** then you can probably skip this section. The parameters needed for proper initial startup include the following:

- **SERVERHOST**

This specifies where Maui will run. It allows Maui client commands to locate the Maui server. It must specify the fully qualified hostname of the machine on which Maui will run. (Example: `SERVERHOST cw.psu.edu`)

- **SERVERPORT**

This specifies the port on which the Maui server will listen for client connections. Unless the default port of 40559 is unacceptable, this parameter need not be set. (Example: `SERVERPORT 50001`)

- **ADMIN1**

Maui has 3 major levels of admin access. Users which are to be granted full control of all Maui functions should be indicated by setting the **ADMIN1** parameter. The first user in this list is considered the *primary* admin. It is the ID under which Maui should always run. Maui will only run under the primary admin user id and will shut itself down otherwise. In order for Maui to properly interact with both PBS and Loadleveler, it is important that the primary Maui admin also be configured as a resource manager admin within each of those systems. (Example: `ADMIN1  joe charles`)

- **RMTYPE[X]**

Maui must be told which resource manager(s) to talk to. Maui currently has interfaces to Loadleveler, Wiki, and PBS. To specify a resource manager, typically only the resource manager type needs to be indicated using the keywords LL, WIKI, or PBS (Example: `RMTYPE[0] PBS`). The array index in the parameter name allows more than one resource manager to be specified. In these multiple resource manager situations, additional parameters may need to be specified depending on the resource manager type. Some of the related resource management parameters are listed below. Further information about each is available in the parameters documentation.

RMPORT
RMSERVER

RMTYPE
RMAUTHTYPE
RMCONFIGFILE

2.3 Initial Maui Testing

Maui 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 Maui in test mode even with your old scheduler running be it an earlier version of Maui or even another scheduler. In test mode, Maui will collect real time job and node information from your resource managers and will act as if it were scheduling live. However, its ability to actually affect jobs (i.e., start, modify, cancel, etc) will be disabled.

Central to Maui testing is the parameter SERVERMODE. This parameter allows administrators to determine how Maui will run. The possible values for this parameter are **NORMAL**, **TEST**, and **SIMULATION**. As would be expected, to request test mode operation, the **SERVERMODE** parameter must be set to **TEST**.

The ultimate goal of testing is to verify proper configuration and operation. Particularly, the following can be checked:

- Maui possesses the minimal configuration required to start up.
- Maui can communicate with the resource manager(s).
- Maui is able to obtain full resource and job information from the resource manager(s).
- Maui is able to properly start a new job

Each of these areas are covered in greater detail below.

2.3.1 Minimal Configuration Required To Start Up

Maui must have a number of parameters specified in order to properly start up. There are three main approaches to setting up Maui on a new system. These include the following:

2.3.1.1 Simulation Mode

Simulation mode is of value if you would simply like to *test drive* the scheduler or when you have a stable production system and you wish to evaluate how or even if the scheduler can improve your current scheduling environment.

An initial *test drive* simulation can be obtained via the following step:

```
> vi maui.cfg
```

(change 'SERVERMODE NORMAL' to 'SERVERMODE SIMULATION')
(add 'SIMRESOURCETRACEFILE traces/Resource.Trace1')
(add 'SIMWORKLOADTRACEFILE traces/Workload.Trace1')

```
> maui &
```

NOTE: In simulation mode, the scheduler does not background itself like it does in both **TEST** and **NORMAL** mode.

The sample workload and resource traces files allow the simulation to emulate a 192 node IBM SP. In this mode, all Maui commands can be run as if on a normal system. The schedctl command can be used to advance the simulation through time. The Simulation chapter describes the use of the simulator in detail.

If you are familiar with Maui, you may wish to use the simulator to tune scheduling policies for your own workload and system. The profiler tool can be used to obtain both resource and workload traces and is described further in the section 'Collecting Traces'. Generally, at least a week's worth of workload should be collected to make the results of the simulation statistically meaningful. Once the traces are collected, the simulation can be started with some initial policy settings. Typically, the scheduler is able to simulate between 10 and 100 minutes of *wallclock* time per second for medium to large systems. As the simulation proceeds, various statistics can be monitored if desired. At any point, the simulation can be ended and the statistics of interest recorded. One or more policies can be modified, the simulation re-run, and the results compared. Once you are satisfied
with the scheduling results, the scheduler can be run *live* with the tuned policies.

2.3.1.2 Test Mode

Test mode allows you to evaluate new versions of the scheduler 'on the side'. In test mode, the scheduler connects to the resource manager(s) and obtains live resource and workload information. Using the policies specified in the maui.cfg file, the test-mode Maui behaves identical to a live 'normal' mode Maui except the code to start, cancel, and pre-empt jobs is disabled. This allows you to exercise all scheduler *code paths* and diagnose the scheduling state using the various diagnostic client commands. The log output can also be evaluated to see if any unexpected states were entered. Test mode can also be used to locate system problems which need to be corrected. Like simulation mode, this mode can also be used to safely *test drive* the scheduler as well as obtain confidence over time of the reliability of the software. Once satisfied, the scheduling mode can be changed from **TEST** to **NORMAL** to begin *live* scheduling.

To set up Maui in test mode, use the following step:

```
> vi maui.cfg
(change 'SERVERMODE NORMAL' to 'SERVERMODE TEST')
> maui
```

Remember that Maui running in test mode will not interfere with your production scheduler, be it Loadleveler, PBS, or even another version of Maui.

NOTE: If you are running multiple versions of Maui, be they in simulation, normal, or test mode, make certain that they each reside in different home directories to prevent conflicts with config and log files, statistics, checkpointing, and *lock* files. Also, each instance of Maui should run using a different SERVERPORT parameter to avoid socket conflicts. Maui client commands can be pointed to the proper Maui server by using the appropriate command line arguments or by setting the environment variable **MAUIHOMEDIR**.

2.3.1.3 Normal Mode

For the adventurous at heart (or if you simply have not yet been properly *burned* by directly installing a large, totally new, mission critical piece of software) or if you are bringing up a new or development system, you may wish to dive in and start the scheduler in **NORMAL** mode. This admin manual and the accompanying man pages should introduce you to the relevant issues and commands. To start the scheduler in **NORMAL** mode, take the following step:

```
> maui
```

That should be all that is needed to get you started.

3.0 Basic Maui Overview

3.1 File Layout

Maui is initially unpacked into a simple one-deep directory structure as shown below. Note that some of the files (i.e., log and statistics files) will be created as Maui is run.

$(MAUIHOMEDIR) _____ **maui.cfg** (general config file containing information required by both the Maui server and user interface clients)
|___ **maui-private.cfg** (config file containing private information required by the Maui server only)
|___ **fs.cfg** (fairshare config file used in Maui 3.0.6 and earlier)
|___ **maui.ck** (Maui checkpoint file)
|___ **maui.pid** (Maui 'lock' file to prevent multiple instances)
|___ **log** (directory for Maui log files - REQUIRED BY DEFAULT)
|____ **maui.log** (Maui log file)
|____ **maui.log.1** (previous 'rolled' Maui log file)
|___ **stats** (directory for Maui statistics files - REQUIRED BY DEFAULT)
|____ Maui stats files (in format 'stats.<YYYY>_<MM>_<DD>')
|____ Maui fairshare data files (in format 'FS.<EPOCHTIME>')
|___ **tools** (directory for local tools called by Maui - OPTIONAL BY DEFAULT)
|___ **traces** (directory for Maui simulation trace files - REQUIRED FOR SIMULATIONS)
|____ resource.trace1 (sample resource trace file)
|____ workload.trace1 (sample workload trace file)
|___ **bin** (directory for Maui executable files - REQUIRED BY DEFAULT)
|____ **maui** (Maui scheduler executable)
|____ **maui_client** (Maui user interface client executable)
|____ **profiler** (tool used to analyze Maui statistics)
|___ **src** (directory for Maui source code files - REQUIRED FOR BUILD)
|___ **spool** (directory for temporary Maui files - REQUIRED FOR ADVANCED FEATURES)
|___ **contrib** (directory containing contributed code in the areas of GUI's, algorithms, policies, etc)

$(MAUIINSTDIR) _____bin (directory for installed Maui executables)

|____ **maui** (Maui scheduler executable)
|____ **maui_client** (Maui user interface client executable)
|____ **profiler** (tool used to analyze Maui statistics)

/etc/maui.cfg (optional file. This file is used to override default '$(MAUIHOMEDIR)' settings. it should contain the string '**MAUIHOMEDIR $(DIRECTORY)**' to override the 'built-in' $(MAUIHOMEDIR)' setting.

When Maui is configured via the **configure** script, the user is queried for the location of the Maui home directory and this directory, $(MAUIHOMEDIR), is compiled in as the default **MAUIHOMEDIR** directory when Maui is built. Unless specified otherwise, Maui will look in this directory for its various config files. If you wish to run Maui out of a different directory, you can override the default home directory setting by creating a **/etc/maui.cfg** file containing the string '**MAUIHOMEDIR <DIRECTORY>**', by setting the environment variable '**MAUIHOMEDIR**', or by specifying the configfile explicitly using the '**-C**' command line option on Maui and the Maui client commands.

When Maui is run, it creates a log file, '**maui.log**' in the log directory and creates a statistics file in the stats directory with the naming convention '**stats.YYYY_MM_DD**' (i.e., 'stats.2000_09_20'). Additionally, a checkpoint file, **maui.ck** and lock file **maui.pid** are maintained in the Maui home directory.

3.2 Scheduling Environment

3.2.1 Scheduling Objects

Maui functions by manipulating five primary, elementary objects. These are jobs, nodes, reservations, QOS structures, and policies. In addition to these, multiple minor elementary objects and composite objects are also utilized. These objects are also defined in the scheduling dictionary.

3.2.1.1 Jobs

Job information is provided to the Maui scheduler from a resource manager such as Loadleveler, PBS, Wiki, or LSF. 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 requirements each of which requests a number of resources of a given type. For example, a job may consist of two requirements, the first asking for '1 IBM SP node with at least 512 MB of RAM' and the second asking for '24 IBM SP nodes with at least 128 MB of RAM'. Each requirements 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 one processor. In SMP environments, however, users may wish to tie one or more processors together with a certain amount of memory and/or other resources.

3.2.1.1.1 Requirement (or Req)

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

- Task Definition

A specification of the elementary resources which compose an individual task.

- Resource Constraints

A specification of conditions which must be met in order for resource *matching* to occur. Only resources from nodes which meet **all** resource constraints may be allocated to the job req.

- Task Count

The number of task instances required by the req.

- Task List

The list of nodes on which the task instances have been located.

- Req Statistics

Statistics tracking resource utilization

3.2.1.2 Nodes

As far as Maui is concerned, a node is a collection of resources with a particular set of associated attributes. In most cases, it fits nicely with the canonical world view of a node such as a PC cluster node or an SP node. In these cases, a node is defined as one or more CPU's, memory, and possibly other compute resources such as local disk, swap, network adapters, software licenses, etc. Additionally, this node will described by various attributes such as an architecture type or operating system. Nodes range in size from small uniprocessor PC's to large SMP systems where a single node may consist of hundreds of CPU's and massive amounts of memory.

Information about nodes is provided to the scheduler chiefly by the resource manager. Attributes include node state, configured and available resources (i.e., processors, memory, swap, etc.), run classes supported, etc.

3.2.1.3 Advance Reservations

An advance reservation is an object which 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 which this access control list will be

enforced. The reservation prevents the listed resources from being used in a way not described by the access control list during the time range specified. For example, a reservation could reserve 20 processors and 10 GB of memory for users Bob and John from Friday 6:00 AM to Saturday 10:00 PM'. Maui uses advance reservations extensively to manage backfill, guarantee resource availability for active jobs, allow service guarantees, support deadlines, and enable metascheduling. Maui also supports both regularly recurring reservations and the creation of dynamic one time reservations for special needs. Advance reservations are described in detail in the advance reservation overview.

3.2.1.4 Policies

Policies are generally specified via a config file and serve to control how and when jobs start. Policies include job prioritization, fairness policies, fairshare configuration policies, and scheduling policies.

3.2.1.5 Resources

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

processors

Processors are specified with a simple count value.

memory

Real memory or 'RAM' is specified in megabytes (MB).

swap

Virtual memory or 'swap' is specified in megabytes (MB).

disk

Local disk is specified in megabytes (MB).

In addition to these elementary resource types, there are two higher level resource concepts used within Maui. These are the *task* and the *processor equivalent*, or PE.

3.2.1.6 Task

A task is a *collection* of elementary resources which must be allocated together within a single node. For example, a task may consist of one processor, 512MB or memory, 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 one node and memory on another.

In Maui, 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 two dual processor nodes.

3.2.1.7 PE

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:

```
   PE = MAX(ProcsRequestedByJob  /
TotalConfiguredProcs,
            MemoryRequestedByJob /
TotalConfiguredMemory,
            DiskRequestedByJob   /
TotalConfiguredDisk,
            SwapRequestedByJob   /
TotalConfiguredSwap) * TotalConfiguredProcs
```

For example, say 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, in this case memory. The processor equivalents for this job should be 50% of the processors, or PE = 64.

Let's make the calculation concrete with one further 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 would be calculated as:

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

This result makes sense since the job would 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 way SMP systems.

3.2.1.8 Class (or Queue)

A class (or queue) is a logical container object which can be used to implicitly or explicitly apply 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 may be associated with a default job duration, default size, or default resource requirements
Host Constraints	A queue may constrain job execution to a particular set of hosts
Job Constraints	A queue may constrain the attributes of jobs which may submitted including setting limits such as max wallclock time, minimum number of processors, etc.
Access List	A queue may constrain who may submit jobs into it based on user lists, group lists, etc.
Special Access	A queue may 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 may be used.

Maui 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* which may be considered to be a ticket to run in a particular class. Any compute node may simultaneously support serveral 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 will consume three batch class initiators on the nodes where it is run.

Using queues as consumable resources allows sites to specify various policies by adjusting the class initiator to node mapping. For example, a site running serial jobs may 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 may set the node's **MAXJOB** policy to 8 and configure the node with 4 special class initiators and 8 batch class initiators.

Note that 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 may choose to associate job

tasks with arbitrary combinations of class initiator requirements.

In displaying class initiator status, Maui 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.

Arbitrary Resource

Node can also be configured to support various 'arbitrary resources'. Information about such resources can be specified using the NODECFG parameter. For example, a node may be configured to have '256 MB RAM, 4 processors, 1 GB Swap, and 2 tape drives'.

3.3 Scheduling Iterations and Job Flow

3.3.1 Scheduling Iterations In any given scheduling iteration, many activities take place. These are broken into the following major categories:

Update State Information
Refresh Reservations
Schedule Reserved Jobs
Schedule Priority Jobs
Backfill Jobs

Update Statistics
Handle User Requests

3.3.1.1 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 which possesses all information.

3.3.1.2 Refresh Reservations

3.3.1.3 Schedule Reserved Jobs

3.3.1.4 Schedule Priority Jobs

In scheduling jobs, multiple steps occur.

3.3.1.5 Backfill Jobs

3.3.1.6 Update Statistics

3.3.1.7 Handle User Requests

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

3.3.1.8 Perform Next Scheduling Cycle

Maui operates on a polling/event driven basis. When all scheduling activities are complete, Maui will process 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 admin 'schedule' requests, reservation activation/deactivation, or the expiration of the RMPOLLINTERVAL timer.

3.3.2 Detailed Job Flow

3.3.2.1 Determine Basic Job Feasibility

The first step in scheduling is determining which jobs are feasible. This step eliminates jobs which have job holds in place, invalid job states (i.e., Completed, Not Queued, Defered, etc), or unsatisfied preconditions. Preconditions may include stage-in files or completion of preliminary job steps.

3.3.2.2 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, length of time the job has been queued, and so forth.

3.3.2.3 Enforce Configured Throttling Policies

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

3.3.2.4 Determine Resource Availability

For each job, Maui attempts to locate the required compute resources needed by the job. In order 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, Maui determine a node to have 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 RESOURCEAVAILABILITYPOLICY parameter can be modified to adjust this behavior.

3.3.2.5 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/or maximize the freedom of the scheduler in making future scheduling decisions.

3.3.2.6 Distribute Jobs Tasks Across Allocated Resources

With the resources selected, Maui then maps job tasks to the actual resources. This distribution of tasks is typically based on simple task distribution algorithms such as round-robin or max blocking, but can also incorporate parallel language library (i.e., MPI, PVM, etc) specific patterns used to minimize interprocesses communication overhead.

3.3.2.7 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.

3.4 Configuring the Scheduler

Maui is configured using the flat text configfile **maui.cfg**. In Maui 3.0.6 and earlier, an optional configfile, **fs.cfg**, could also be specified to define fairshare and QoS configuration. In more recent versions, this functionality is handled by the ***CFG** parameters within the **maui.cfg** file.

All config files consist of simple '<PARAMETER> <VALUE>' pairs which 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>]', i.e., 'QOSCFG[hiprio] PRIORITY=1000' The <VALUE> settings may be integers, floats, strings, or arrays of these. Some parameters can be specified as arrays and in such, index values can be numeric or alphanumeric strings. If no array index is specified for an array parameter, an index of '0' is assumed. See the parameters documentation for information on specific parameters.

All config files are read when Maui is started up. Also, the **schedctl -R** command can be used to reconfigure the scheduler at any time, forcing it to re-read all config files before continuing. The command changeparam can be used to change individual parameter settings at any time, i.e. '`changeparam LOGLEVEL 3`'. Changes made by the changeparam command are not persistent so will be overwritten the next time the config file values are loaded. The current parameter settings can be viewed at any time using the showconfig command.

4.0 Maui Commands

4.1 Client Overview

4.2 Status Commands

Maui provides an array of commands to organize and present information about the current state and historical statistics of the scheduler, jobs, resources, users, accounts, etc. The table below presents the primary status commands and flags. The Command Overview lists all available commands.

Command	Flags	Description
checkjob		display job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization
checknode		display node state, resources, attributes, reservations, history, and statistics
diagnose	-j	display summarized job information and any unexpected state
diagnose	-n	display summarized node information and any unexpected state
showgrid		display various aspects of scheduling performance across a job duration/job size matrix
showq	[-r | -i]	display various views of currently queued active, idle, and non-eligible jobs
showstats	-f	display historical fairshare usage on a per credential basis
showstats	-g	display current and historical usage on a per group basis
showstats	-u	display current and historical usage on a per user basis
showstats	-v	display high level current and historical scheduling statistics

4.3 Job Management Commands

Maui 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, hold state, etc. The table below covers the available job management commands. The Command Overview lists all available commands.

Command	Flags	Description
canceljob		cancel existing job
checkjob		display job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization
diagnose	-j	display summarized job information and any unexpected state
releasehold	[-a]	remove job holds or defers
runjob		start job immediately if possible
sethold		set hold on job
setqos		set/modify QoS of existing job
setspri		adjust job/system priority of job

4.4 Reservation Management Commands

Maui exclusively controls and manages all advance reservation features including both standing and administrative reservations. The table below covers the available reservation management commands. The Command Overview lists all available commands.

Command	Flags	Description
diagnose	-r	display summarized reservation information and any unexpected state
releaseres		remove reservations
setres		immediately create an administrative reservation
showres		display information regarding location and state of reservations

4.5 Policy/Config Management Commands

Maui allows dynamic modification of most scheduling parameters allowing new scheduling policies, algorithms, constraints, and permissions to be set at any time. Changes made via Maui client commands are temporary and will be overridden by values specified in Maui config files the next time Maui is shutdown and restarted. The table below covers the available configuration management commands. The Command Overview lists all available commands.

Command	Flags	Description
changeparam		immediately change parameter value
schedctl		control scheduling behavior (i.e., stop/start scheduling, recycle, shutdown, etc.)
showconfig		display settings of all configuration parameters

4.6 End User Commands

While the majority of Maui commands are tailored for use by system administrators, a number of commands are designed to extend the knowledge and capabilities of end users. The table below covers the commands available to end users. The Command Overview lists all available commands.

Command	Flags	Description
canceljob		cancel existing job
checkjob		display job state, resource requirements, environment, constraints, credentials, history, allocated resources, and resource utilization
showbf		show resource availability for jobs with specific resource requirements
showq		display detailed prioritized list of active and idle jobs
showstart		show estimated start time of idle jobs
showstats		show detailed usage statistics for users, groups, and accounts which the end user has access to

4.7 Miscellaneous Commands

The table below covers a number of additional commands which do not fully fit in prior categories. The Command Overview lists all available commands.

Command	Flags	Description
resetstats		reset internal statistics

5.0 Assigning Value - Job and Resource Prioritization

5.1 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 which 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 Maui in representing a multi-facetted 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.

5.1.1 Priority Overview

Maui's prioritization mechanism allows component and subcomponent weights to be associated with many aspects of a job so as to enable fine-grained control over this aspect of scheduling. To allow this level of control, Maui uses a simple priority-weighting hierarchy where the contribution of each priority subcomponent is calculated as

<COMPONENT WEIGHT> * <SUBCOMPONENT WEIGHT> * <PRIORITY SUBCOMPONENT VALUE>

Each priority component contains one or more subcomponents as described in the Priority Component Overview. For example, the Resource component consists of Node, Processor, Memory, Swap, Disk, and PE subcomponents. While there are numerous priority components and many more subcomponents, a site need only focus on and configure the subset of components related to their particular priority needs. In actual usage, few sites use more than a small fraction (usually 5 or less) of the available priority subcomponents. This results in fairly straightforward priority configurations and tuning. By mixing and matching priority weights, sites may generally obtain the desired job-start behavior. At any time, the diagnose -p command can be issued to determine the impact of the current priority-weight settings on idle jobs. Likewise, the command showgrid 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 which is set to 1. This results in a total job priority equal to the period of time the job has been queued, causing Maui 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.

Using the setspri command, site admins may adjust the base calculated job priority by either assigning a relative priority adjust or an absolute system priority. A relative priority adjustment will cause the base priority to be increased or decreased by a specified value. Setting an absolute system priority, SPRIO, will cause the job to receive a priority equal to MAX_PRIO_VAL + SPRIO, and thus guaranteed to be of higher value than any naturally occurring job priority.

5.1.2 Job Priority Factors

Maui allows jobs to be prioritized based on a range of job related factors. These factors are broken down into a two-level hierarchy of priority factors and subfactors each of which can be independently assigned a weight. This approach provides the administrator with detailed yet straightforward control of the job selection process. The table below highlights the components and subcomponents which make up the total job priority.

Component	SubComponent	Metric
CRED (job credentials)	USER	user specific priority (See USERCFG)
	GROUP	group specific priority (See GROUPCFG)
	ACCOUNT	account specific priority (SEE ACCOUNTCFG)
	QOS	QOS specific priority (See QOSCFG)
	CLASS	class/queue specific priority (See CLASSCFG)
FS (fairshare usage)	FSUSER	user based historical usage (See Fairshare Overview)
	FSGROUP	group based historical usage (See Fairshare Overview)
	FSACCOUNT	account based historical usage (See Fairshare Overview)
	FSQOS	QOS base historical usage (See Fairshare Overview)
	FSCLASS	class/queue based historical usage (See Fairshare Overview)
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 proc-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
TARGET (target service levels)	TARGETQUEUETIME	time until queuetime target is reached (exponential)
	TARGETXFACTOR	distance to target expansion factor (exponential)
USAGE (consumed resources -- active jobs only)	CONSUMED	proc-seconds dedicated to date
	REMAINING	proc-seconds outstanding
	PERCENT	percent of required walltime consumed

5.1.2.1 Credential (CRED) Component

The credential component allows a site 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:

```
Priority += CREDWEIGHT * (
    USERWEIGHT    * J->U->Priority +
    GROUPWEIGHT   * J->G->Priority +
    ACCOUNTWEIGHT * J->A->Priority +
    QOSWEIGHT     * J->Q->Priority +
    CLASSWEIGHT   * J->C->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, the following might be used.

```
---
CREDWEIGHT      1
USERWEIGHT      1
GROUPWEIGHT     1

USERCFG[john]   PRIORITY=2000
USERCFG[paul]   PRIORITY=-1000

GROUPCFG[staff] PRIORITY=10000
---
```

 Class (or queue) priority may also be specified via the resource manager where supported (i.e., PBS queue priorities). However, if Maui class priority values are also specified, the resource manager priority values will be overwritten.

All priorities may be positive or negative.

5.1.2.2 Fairshare (FS) Component

Fairshare components allow a site to favor jobs based on short term historical usage. The Fairshare Overview describes the configuration and use of Fairshare in detail.

After the brief reprieve from complexity found in the QOS factor, we come to the Fairshare factor. This factor is used to adjust a job's priority based on the historical percentage system utilization of the jobs user, group, account, or QOS. This allows you to 'steer' the workload toward a particular usage mix across user, group, account, and QOS dimensions. The fairshare priority factor calculation is

```
Priority += FSWEIGHT * MIN(FSCAP, (
    FSUSERWEIGHT     * DeltaUserFSUsage +
    FSGROUPWEIGHT    * DeltaGroupFSUsage +
    FSACCOUNTWEIGHT  * DeltaAccountFSUsage +
    FSQOSWEIGHT      * DeltaQOSFSUsage +
    FSCLASSWEIGHT    * DeltaClassFSUsage))
```

All '***WEIGHT**' parameters above are specified on a per partition basis in the maui.cfg file. The 'Delta*Usage' components represents the difference in actual fairshare usage from a fairshare usage target. Actual fairshare usage is determined based on historical usage over the timeframe specified in the fairshare configuration. The target usage can be either a target, floor, or ceiling value as specified in the fairshare config file. The fairshare documentation covers this in detail but an example should help obfuscate things completely. 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 3
  Fairshare Target: 10.0+
  Current Fairshare Usage: 25.0

Class E
  Fairshare Target: [NONE]
  Current Fairshare Usage: 20.0

PriorityWeights:
  FSWEIGHT 100
  FSUSERWEIGHT 10
  FSGROUPWEIGHT 20
  FSACCOUNTWEIGHT 30
  FSQOSWEIGHT 40
  FSCLASSWEIGHT 0

In this example, the Fairshare component calculation would be as follows:

```
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 3 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. Thus, the QOS 3 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%. See the Fairshare Overview for more information.

5.1.2.3 Resource (RES) Component

Weighting jobs by the amount of resources requested allows a site to favor particular types of jobs. Such prioritization may allow a site 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; first, when you need to favor large resource jobs because its part of your site's mission statement; second, when you want to level the response time distribution across large and small jobs (small jobs are more easily backfilled and thus generally have better turnaround time); and finally, when you want to improve system utilization. What? Yes, 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 thus increase overall system utilization. Its a lot like the story about filling a cup with golf balls and sand. If you put the sand in first, it gets in the way when you try to put in the 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:

```
Priority += RESWEIGHT * MIN(RESOURCECAP, (
    NODEWEIGHT     * TotalNodesRequested +
    PROCWEIGHT     * TotalProcessorsRequested +
    MEMWEIGHT      * TotalMemoryRequested +
    SWAPWEIGHT     * TotalSwapRequested +
    DISKWEIGHT     * TotalDiskRequested +
    PEWEIGHT       * TotalPERequested))
```

The sum of all weighted resources components is then multiplied by the **RESWEIGHT** parameter and capped by the **RESOURCECAP** 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 O2K system, it would have a PE value of MAX(25,50) * 128, or 64. The concept of PE's may be a little awkward to grasp initially but it is a highly effective metric in shared resource systems.

5.1.2.4 Service (SERV) Component

The Service component essentially specifies which service metrics are of greatest value to the site. Favoring one service subcomponent over another will generally cause that service metric to improve.

5.1.2.4.1 QueueTime (QUEUETIME) Subcomponent

In the priority calculation, a job's queue time is a duration measured in minutes. Use of 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 Maui. It is important to note that within Maui, a job's queue time is not necessarily the amount of time since the job was submitted. The parameter

JOBPRIOACCRUALPOLICY allows a site 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 most sites have pretty smart users and pretty smart users 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 'pre submitted' 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 as well as preventing 'queue stuffers' from doing similar things on a shorter time scale. These 'queue stuffer' users submit hundreds of jobs at once so as to swamp the machine and hog 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/or active Maui fairness policies. (i.e., max job per user, max idle job per user, etc.)

As a final note, the parameter QUEUETIMEWEIGHT can be adjusted on a per QOS basis using the QOSCFG parameter and the **QTWEIGHT** attribute. For example, the line '`QOSCFG[special] QTWEIGHT=5000`' will cause jobs utilizing the QOS `special` to have their queue time subcomponent weight *increased* by 5000.

5.1.2.4.2 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 canonical form, the expansion factor (XFactor) metric is calculated as

```
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, 'Execution Time', 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, Maui does not necessarily use the *raw* time since job submission to determine 'QueueTime' so as to prevent various scheduler abuses. Consequently, Maui uses the following modified equation:

```
XFACTOR = 1 + <EFFQUEUETIME> / <WALLCLOCKLIMIT>
```

In the equation above, `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 table below 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	(2 + 1) / 1 = 3.00	(4 + 1) / 1 = 5.00	(8 + 1) / 1 = 9.00	(16 + 1) / 1 = 17.0
XFactor for 4 hour job	(1 + 4) / 4 = 1.25	(2 + 4) / 4 = 1.50	(4 + 4) / 4 = 2.00	(8 + 4) / 4 = 3.00	(16 + 4) / 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 XFCAP, or by using the XFMINWCLIMIT parameter. If the later is used, the calculation for the xfactor subcomponent value becomes:

```
 XFACTOR = 1 + <EFFQUEUETIME> /
MAX(<XFMINWCLIMIT>,<WALLCLOCKLIMIT>)
```

The use of the **XFMINWCLIMIT** parameter allows a site 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 consequently, job XFactor distribution tends to be fairly level across a wide range of job durations. (i.e., A flat XFactor distribution of 1.0 would result in a one minute job being queued on average one minute, while a 24 hour job would be queued an average of 24 hours).

Like queue time, the effective XFactor subcomponent weight is the sum of two weights, the **XFWEIGHT** parameter and the QOS specific XFWEIGHT setting. For example, the line '`QOSCFG[special] XFWEIGHT=5000`' will cause jobs utilizing the QOS `special` to have their expansion factor subcomponent weight *increased* by 5000.

5.1.2.4.3 Bypass (BYPASS) Subcomponent

The bypass factor is the forgotten stepchild of the priority subcomponent family. It was originally introduced to prevent backfill based starvation. It 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. The calculation for this factor is simply. Over the years, the anticipated backfill starvation has never been reported. The good news is that if it ever shows up, Maui is ready!

5.1.2.5 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 which 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:

```
Priority += TARGWEIGHT * (
    QueueTimeComponent +
    XFactorComponent)
```

The queue time and expansion factor target are specified on a per QOS basis using the 'QOSXFTARGET' and 'QOSQTTARGET' parameters. The QueueTime and XFactor component calculations are designed produce small values until the target value begins to approach at which point these components grow very rapidly. If the target is missed, these component will remain high and continue to grow but will not grow exponentially.

5.1.2.6 Usage (USAGE) Component

(Under Construction)

5.1.3 Common Priority Usage

Sites vary wildly in the preferred manner of prioritizing jobs. Maui's scheduling hierarchy allows sites to meet their job control needs without requiring them to adjust dozens of parameters. Some sites may choose to utilize numerous subcomponents, others a few, and still others are completely happy with the default FIFO behavior. Any subcomponent which is not of interest may be safely ignored.

To help clarify the use of priority weights, a brief example may help. Suppose a site wished to maintain the FIFO behavior but also incorporate some credential based prioritization to favor a special user. Particularly, the site would like the userjohn to receive a higher initial priority than all other users. Configuring this behavior would require two steps. First, the user credential subcomponent would need to be enabled and second, john would need to have his relative priority specified. Take a look at the example maui.cfg:

```
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 (i.e., **RESWEIGHT**, **TARGETQUEUETIMEWEIGHT**, etc.).

The second part of the example involved specifying the actual user priority for the user john. This was 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 (see table above), the user priority of 300 will make 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. The honest truth is that most sites are not completely certain what they want in prioritization at the onset. Most often, prioritization is a tuning process where an *initial stab* is made and adjustments are then made over time. Unless you are an exceptionally stable site, prioritization is also not a matter of getting it right. 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 on.

Lets look at one more example. A site wants to do the following:

- favor jobs in the low, medium, and high QOS's so they will run in QOS order
- balance job expansion factor
- use job queue time to prevent jobs from starving

The sample **maui.cfg** is listed below:

```
---
QOSWEIGHT               1
XFACTORWEIGHT           1
QUEUETIMEWEIGHT         10
TARGETQUEUETIMEWEIGHT   1

QOSCFG[low]         PRIORITY=1000
QOSCFG[medium]      PRIORITY=10000
QOSCFG[high]        PRIORITY=10000
USERCFG[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, **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.

Assume now that the site decided that it liked this priority mix but they had a problem with users 'cheating' by submitting large numbers very short jobs. They would do this because very short jobs would tend to have rapidly growing xfactor values and would consequently quickly jump to the head of the queue. In this case, a 'factor cap' would be appropriate. These caps allow a site to say I would like this priority factor to contribute to a job's priority but only 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 (i.e., **QUEUETIMECAP**, **RESCAP**, **SERVCAP**, etc.) Note that both component and subcomponent caps apply to the 'pre-weighted' 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) +
    ...)) +
```

. . .

5.1.4 Prioritization Strategies

Each component or subcomponent may 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 which have been waiting the longest.

- **Queue Time**
- **Expansion Factor**
- **Resource**
- **Fairshare**
- **Cred**
- **Target Metrics**

Each priority factor group may contain one or more subfactors. For example, the Resource factor consists of Node, Processor, Memory, Swap, Disk, and PE components. Figure <X> shows the current priority breakdown. From the figure, it is quickly apparent that the prioritization problem is fairly 'nasty' due to the fact that every site needs to prioritize a bit differently. Fortunately, there has not yet been a site that has desired to use more than a fraction of these priority factors, thus greatly simplifying the job priority tuning issue. 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).

Each priority factor is reviewed in detail below. The command 'diagnose -p' is designed to assist in visualizing the priority distribution resulting from the current job priority configuration. Also, the showgrid command will help indicate the impact of the current priority settings on scheduler service distributions.

5.1.5 Manual Job Priority Adjustment

Batch administrator's regularly find a need to adjust the calculated priority of a job to meet current needs. Current needs often are broken into two categories:

A) The need to run an admin test job as soon as possible
B) The need to pacify an irate user

Under Maui, the setspri command can be used to handle these issues in one of two ways. This command allows the specification of either a relative priority adjustment, or the specification of a absolute priority. Using absolute priority specification, administrators can set a job priority which is guaranteed to be higher than any calculated value. Where Maui-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 command 'setspri <PRIO> <JOBID>', for example, will assign a priority of '1 billion + <PRIO>' to the job. Thus, 'setspri 5 job.1294' with set the priority of job 'job.1294' to 1000000005.

5.2 Node Allocation

While job prioritization allows a site to determine which job to run, node allocation policies allow a site 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, available resource constraints, and other issues. The following algorithms are available and described in detail below: FIRSTAVAILABLE, LASTAVAILABLE, MACHINEPRIO, CPULOAD, MINRESOURCE, CONTIGUOUS, MAXBALANCE, FASTEST, and LOCAL. Additional load allocation polices such as may be enabled through extension libraries such as G2. Documentation for the extension library of interest should be consulted.

5.2.1 Node Allocation Overview

Node allocation is important in the following situations:

- heterogeneous system

If the available compute resources have differing configurations, and a subset of the submitted jobs cannot run on all of the nodes, then allocation decisions can significantly affect scheduling performance. For example, a system may be comprised of two nodes, A and B, which are identical in all respects except for RAM, possessing 256MB and 1GB of RAM respectively. Two single processor jobs, X and Y, are submitted, one requesting at least 512 MB of RAM, the other, at least 128 MB. The scheduler could run job X on node A in which case job Y would be blocked until job X completes. A more intelligent approach may be to allocate node B to job X because it has the fewest available resources yet still meets the constraints. This is somewhat of a 'bestfit' approach in the configured resource dimension and is essentially what is done by the 'MINRESOURCE' algorithm.

- shared node system

Shared node systems are most often involve SMP nodes although this is not mandatory. Regardless, when sharing the resources of a given node amongst tasks from more than one job, resource contention and fragmentation issues arise.

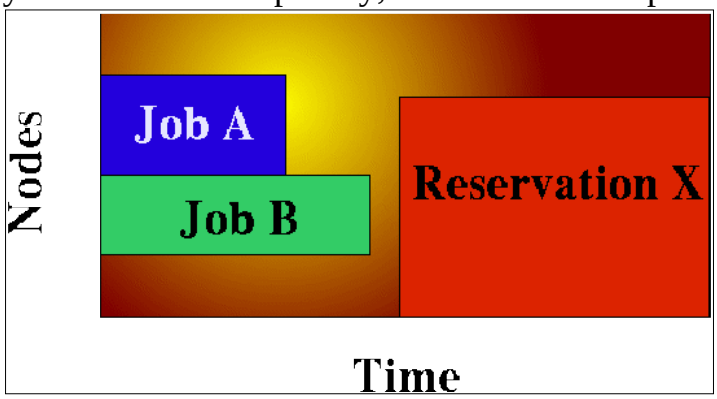
Most current systems still do not do a very good job of logically partitioning the resources (i.e., CPU, Memory, network bandwidth, etc.) available on a given node. Consequently contention often arises between tasks of independent jobs on the node. This can result in a slowdown for all jobs involved which can have significant ramifications if large way parallel jobs are involved.

On large way SMP systems (i.e., > 32 processors/node), job packing can result in intra-node fragmentation. For example, again take two nodes, A and B each with 64 processors. Assume they are currently loaded with various jobs and have 24 and 12 processors free respectively. Two jobs are submitted, Job X requesting 10 processors, and job Y requesting 20 processors. Job X can start on either node but starting it on node A will prevent job Y from running. An algorithm to handle intra-node fragmentation is pretty straightforward for a single resource case, but what happens when jobs request a combination of processors, memory, and local disk. Determining the correct node suddenly gets significantly more difficult. Algorithms to handle these type of issues are currently available in the G2 extension library.

- reservation based systems

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 top minimize this fragmentation and maximize the schedulers ability to continue to start jobs in existing slots. See the figure to hopefully remove a small amount of the incoherency contained in the above sentences. In this figure,

Job A and job B are already running. A reservation, X, has been 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. Hopefully, this canned 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 which have the smallest space between the end of a job under consideration and the start of a future reservation.



- non-flat network system

On systems where network connections do not resemble a flat 'all-to-all' topology, the placement of tasks may present a significant impact on the performance of communication intensive parallel jobs. If latencies and bandwidth of the network 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 the impact of these bandwidth and latency differences.

5.2.2 Resource Based Algorithms

Maui contains a number of allocation algorithms which address some of the needs described above. Additional 'homegrown' allocation algorithms may also be created and interfaced into the Maui scheduling system. The current suite of algorithms is described below.

5.2.2.1 CPULOAD

Nodes are selected which have the maximum amount of available, unused cpu power, i.e. <#of CPU's> - <CPU load>. Good algorithm for timesharing node systems. This algorithm is only applied to jobs starting immediately. For the purpose of future reservations, the MINRESOURCE algorithm is used.

5.2.2.2 FIRSTAVAILABLE

Simple first come, first server algorithm where nodes are allocated in the order they are presented by the resource manager. This is a very simple, and very fast algorithm.

5.2.2.3 LASTAVAILABLE

Algorithm which selects resources so as to minimize the amount of time after the job and before the the trailing reservation. This algorithm is a 'best fit in time' algorithm which minimizes the impact of reservation based node-time fragmentation. It is useful in systems where a large number of reservations (job, standing, or administrative) are in place.

5.2.2.4 MACHINEPRIO

This algorithm allows a site to specify the priority of various static and dynamic aspects of compute nodes and allocate them accordingly. It is in essence a flexible version of the **MINRESOURCE** algorithm.

5.2.2.5 MINRESOURCE

This algorithm priorities nodes according to the configured resources on each node. Those nodes with the fewest configured resources which still meet the job's resource constraints are selected.

5.2.2.6 CONTIGUOUS

This algorithm will allocate nodes in contiguous (linear) blocks as required by the Compaq RMS system.

5.2.2.7 MAXBALANCE

This algorithm will attempt 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 speed. Thus, if possible, nodes with identical speeds will be allocated to the job. If identical speed nodes cannot be found, the algorithm will allocate the set of nodes with the minimum node speed 'span' or range.

5.2.2.8 FASTEST

This algorithm will select nodes in 'fastest node first' order. Nodes will be selected by node *speed* if specified. If node speed is not specified, nodes will be selected by processor speed. If neither is specified, nodes will be selected in a random order.

5.2.2.9 LOCAL

This will call the locally created 'contrib' node allocation algorithm.

See also

N/A.

5.2.3 Time Based Algorithms

Under Construction

5.2.4 Locally Defined Algorithms

Under Construction

6.0 Managing Fairness - Throttling Policies, Fairshare, and Allocation Management

---

6.1 Fairness Overview

The concept of *fairness* varies widely from person to person and site to site. To some it implies giving all users equal access to compute resources. However, more complicated concepts incorporating historical resource usage, political issues, and job *value* are equally valid. While no scheduler can handle all possible definitions of what *fair* means, Maui provides some flexible tools that help with most common fairness management definitions and needs. Particularly, fairness under Maui may be addressed by any combination of the facilities described in the table below.

Facility	Description	Example

Thottling Policies	Specify limits on exactly what resources can be used at any given instant.	`USERCFG[john] MAXJOB=3`
		`GROUPCFG[DEFAULT] MAXPROC=64`
		`GROUPCFG[staff]`
		`MAXPROC=128`
		
		(allow `john` to only run 3 jobs at a time. Allow the group `staff` to utilize up to 128 total processors and all other groups to utilize up to 64 processors.)

Job Prioritization	Specify what is most important to the scheduler. Using *Service* based priority factors can allow a site to balance job turnaround time, expansion factor, or other scheduling performance metrics.	`SERVWEIGHT 1`
		`QUEUETIMEWEIGHT 10`
		
		(cause jobs to increase in priority by 10 points for every minute they remain in the queue.)

Fairshare	Specify usage targets to limits resource access or adjust priority based on historical resource usage.	`USERCFG[steve]`
		`FSTARGET=25.0+`
		`FSWEIGHT 1`
		`FSUSERWEIGHT 10`
		
		(enable *priority based* fairshare and specify a fairshare target for user `steve` such that his job's will be favored in an attempt to keep his job's utilizing *at least* 25.0% of delivered compute cycles.)

Allocation Management	Specify long term, credential-based resource usage limits.	`BANKTYPE QBANK`
		`BANKSERVER server.sys.net`
		
		(enable the **QBank** allocation management system. Within the allocation manager, project or account based allocations may be configured. These allocations may, for example, allow project **X** to utilize up to 100,000 processor-hours per quarter, provide various QoS sensitive charge rates, share allocation access, etc.)

6.2 Throttling Policies

Maui possesses a number of policies which allow an administrator to control the flow of jobs 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 may be specified as global or specific constraints specified on a per user, group, account, QOS, or class basis.

6.2.1 Fairness via Throttling Policies

Significant improvements in the flexibility of throttling policies were introduced in Maui 3.0.7. Those sites using versions prior to this should consult the Maui 3.0.6 style throttling policy configuration documentation. At a high level, Maui allows resource usage limits to be specified for in three primary dimensions:

6.2.1.1 Basic Fairness Policies

- Active Job Limits (Constrains the total cumulative resource available to active jobs at a given time)
- Idle Job Limits (Constrains the total cumulative resources available to idle jobs at a given time)
- System Job Limits (Constrains 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. Additionally, QoS's may be configured to allow limit overrides to any particular policy. For a job to run, it must meet all policy limits. Limits are applied using the '*CFG' set of parameters, particularly, USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, CLASSCFG, and SYSTEMCFG. Limits are specified by associating the desired limit to the individual or default object. The usage limits currently supported by Maui listed

in the table below.

NAME	UNITS	DESCRIPTION	EXAMPLE
MAXJOB	# of jobs	Limits the number of jobs a credential may have active (Starting or Running) at any given time.	MAXJOB=8
MAXPROC	# of processors	Limits the total number of dedicated processors which can be allocated by active jobs at any given time.	MAXPROC=32
MAXPS	<# of processors> * <walltime>	Limits the number of outstanding processor-seconds a credential may have allocated at any given time. For example, if a user has a 4 processor job which will complete in 1 hour and a 2 processor job which will complete in 6 hours, he has '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 job's approach their completion time.	MAXPS=720000

MAXPE	# of processor equivalents	Limits the total number of dedicated processor-equivalents which can be allocated by active jobs at any given time.	MAXPE=128
MAXWC	job duration [[[DDD:]HH:]MM:]SS	Limits the number of outstanding seconds a credential may have associated with active jobs. It behaves identically to the MAXPS limit above only lacking the processor weighting. Like MAXPS, the outstanding second usage of each credential is also updated each scheduling iteration.	MAXWC=72:00:00
MAXNODE	# of nodes	limits the total number of compute nodes which can be in use by active jobs at any given time.	MAXNODE=64
MAXMEM	total memory in MB	Limits the total amount of dedicated memory (in MB) which can be allocated by a credential's active jobs at any given time.	MAXMEM=2048

The example below demonstrates a simple limit specification.

```
---
USERCFG[DEFAULT]   MAXJOB=4
USERCFG[john]      MAXJOB=8
---
```

This example will allow user john to run up to 8 jobs while all other users may only run up to 4.

Simultaneous limits of different types may be applied per credential and multiple types of credential may have limits specified. The next 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. User steve limits will cause that jobs submitted under his user ID will be constrained so that he may only run up to 2 simultaneous jobs with an aggregate node consumption of 30 nodes. However, if he submits a job to a class other than batch, he may be limited further. Only 16 total nodes may 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.

6.2.1.2 Multi-Dimension Fairness Policies

Multi-dimensional fairness policies allow a site 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. Early versions of Maui 3.2 enabled the following multi-dimensional fairness policies:

MAXJOB[Class,User]
MAXNODE[Class,User]
MAXPROC[Class,User]

These limits would be specified in the following manner:

```
CLASSCFG[X]  MAXJOBPERUSER=<LIMIT>
CLASSCFG[X]  MAXNODEPERUSER=<LIMIT>
CLASSCFG[X]  MAXPROCPERUSER=<LIMIT>
```

Later versions of Maui will allow more generalized use of these limits using the following syntax:

{<O1>}CFG[<OID1>] MAX{<A1>}[[<O2>[:<OID2>]]]=<LIMIT>

Where

O1 is one of the following objects: **USER**, **GROUP**, **ACCOUNT**, **QOS**, or **CLASS**

A1 is one of the following attributes: **JOB**, **PROC**, **PS**, **PE**, **WC**, **NODE**, or **MEM**
O2 is one of the following objects: **USER**, **GROUP**, **ACCOUNT**, **QOS**, or **CLASS**

If OID2 is specified, the limit is applied only to that object instance. Otherwise, the limit is applied to all appropriate objects by default.

The following examples may clarify:

```
CLASSCFG[batch] MAXJOB=3 MAXNODE[USER]=8
```

Allow class `batch` to run up the 3 simultaneous jobs. Allow any user to use up to 8 total nodes within class `batch`.

```
CLASSCFG[fast] MAXPROC[USER:steve]=3
MAXPROC[USER:bob]=4
```

Allow users `steve` and `bob` to use up to 3 and 4 total processors respectively within class `fast`.

See Also:

N/A

6.2.2 Override Limits

Like all job credentials, the QOS object may be also be associated with resource usage limits. However, this credential can also be given special override limits which supersede the limits of other credentials. Override limits are applied by preceding the limit specification with the letter 'O'. The configuration below provides an example of this.

```
---
USERCFG[steve]    MAXJOB=2  MAXNODE=30
GROUPCFG[staff]   MAXJOB=5
CLASSCFG[DEFAULT] MAXNODE=16
CLASSCFG[batch]   MAXNODE=32
QOSCFG[hiprio]    MAXJOB=3  OMAXNODE=64
----
```

This configuration is identical to the line above with the exception of the final QOSCFG line. This line does two things

- Only 3 highprio jobs may run simultaneously
- highprio QOS jobs may run with up to 64 nodes per credential ignoring other credential MAXNODE limits.

Given the above configuration, assume a job was now submitted with the credentials, user

steve, group staff, class batch, and QOS hiprio.

This job will be allowed to start so long as running it does not lead to any of the following conditions:

- total nodes used by user steve jobs do 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 jobs do not exceed 64.
- total active jobs associated with QOS hiprio does not exceed 3.

While the above example is a bit complicated for actual use at most sites, similar combinations may be needed to enforce site policies on many larger systems.

6.3 FairShare

6.3.1 Overview

Fairshare is a mechanism which allows historical resource utilization information to be incorporated into job feasibility and priority decisions. Maui's fairshare implementation allows site administrators to set system utilization targets for users, groups, accounts, classes, and QOS levels. Administrators can also specify the timeframe over which resource utilization is evaluated in determining whether or not the goal is being reached. This timeframe is indicated by configuring parameters specifying the number and length of fairshare windows which are evaluated to determine historical resource usage. Fairshare targets can then be specified for those credentials (i.e., user, group, class, etc) which administrators wish to have affected by this information.

6.3.2 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. Secondly, some configuration is required on a per credential basis to determine how this fairshare information affects particular jobs. The system level parameters are listed below:

FSINTERVAL - specifies the timeframe covered by each fairshare window.
FSDEPTH - specifies the number of windows to be evaluated when determining current fairshare utilization.
FSDECAY - specifies the decay factor to be applied to fairshare windows.
FSPOLICY - specifies the metric to use when tracking fairshare usage (if set to NONE, fairshare information will not be used for either job prioritization or job feasibility evaluation)
FSCONFIGFILE - specifies the name of the file which contains the per user, group, account, and QOS fairshare configuration. (fs.cfg by default)

In earlier versions of Maui (Maui 3.0.6 and earlier) fairshare configuration information was specified via the fs.cfg file. In Maui 3.0.7 and higher, although use of the fs.cfg file is still supported, it is recommended that the *CFG suite of parameters (ACCOUNTCFG, CLASSCFG, GROUPCFG, QOSCFG, and USERCFG) be used. Both approaches allow

specification of per user, group, account, and QOS fairshare in terms of target limits and target types.

As Maui runs, it records how available resources are being utilized. Each iteration (RMPOLLINTERVAL seconds) it updates fairshare resource utilization statistics. Currently, resource utilization is measured in accordance with the FSPOLICY parameter allowing various aspects of resource consumption information to be tracked. 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 while consumed usage tracks what the job actually uses. An example may clarify this. Assume a 4 processor job is running a parallel '/bin/sleep' for 15 minutes. It will have a dedicated fairshare usage of 1 proc-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.

Using the selected fairshare usage metric, Maui 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 Maui statistics directory. Each file is named 'FS.<EPOCHTIME>' where <EPOCHTIME> is the time the new fairshare window became active. Each window contains utilization information for each entity as well as for total usage. A sample fairshare data file is shown below:

```
-----
# Fairshare Data File (Duration: 172800 Seconds)  Starting:
Fri Aug 18 18:00:00

User            USERA   150000.000
User            USERB   150000.000
User            USERC   200000.000
User            USERD   100000.000
Group           GROUPA  350000.000
Group           GROUPB  250000.000
Account         ACCTA   300000.000
Account         ACCTB   200000.000
Account         ACCTC   100000.000
QOS             0       50000.000
QOS             1       450000.000
QOS             2       100000.000
TOTAL                   600000.00
-----
```

Note that the total processor hours consumed in this time interval is 600,000 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 (i.e., USERA + USERB + ... + USERD = GROUPA + GROUPB = ACCTA + ...

+ ACCTC = QOS0 + ... + QOS2 = TOTAL)

When Maui needs to determine current fairshare usage for a particular entity, it performs a 'decay-weighted' average the usage information for that entity contained in the FSDEPTH most recent windows. For example, assume the entity of interest is user John and the following parameters are set,

```
FSINTERVAL        12:00:00
FSDEPTH                 4
FSDECAY               0.5
```

and the fairshare data files contain the following usage amounts for the entity of interest:

```
John[0]         60.0
Total[0]        110.0

John[1]         0.0
Total[1]        125.0

John[2]         10.0
Total[2]        100.0

John[3]         50.0
Total[3]        150.0
```

The current fairshare usage for user John would calculated as follows:

Usage = (60 + .5^1 * 0 + .5^2 * 10 + .5^3 * 50) / (110 + .5^1*125 + .5^2*100 + .5^3*150)

Note that 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.

When configuring fairshare, it is important to determine the proper timeframe that should be considered. Many sites choose one to two weeks to be the total timeframe covered (i.e., **FSDEPTH** * **FSINTERVAL**) but any reasonable timeframe should work. How this timeframe is broken up between the number and length of windows is a matter of preference, just note that more windows means that the decay factor will make aged data less significant more quickly.

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 may result in unexpected fairshare status for a period of time as the fairshare data files with the old **FSINTERVAL** setting are rolled out.

6.3.3 Using Fairshare Information

With the mechanism used to determine current fairshare usage explained above, we can now move on that actually using this information. As mentioned in the fairshare overview, fairshare information primarily used in determining the fairshare priority factor. This factor is actually calculated by determining the difference between the actual fairshare usage of an entity and a specified target usage.

See Also:

The 'diagnose -f' command was created to allow diagnosis and monitoring of the fairshare facility.

FairShare Prioritization vs Hard FairShare Enforcement

FSENFORCEMENT

6.4 Allocation Management

Overview

An allocations manager (also known as an allocations bank or cpu bank) is a software system which manages resource allocations where a resource allocation grants a job a right to use a particular amount of resources. This is not the right place for a full allocations manager overview but a brief review may point out the value in using such a system.

An allocations manager functions much like a bank in that it provides a form a currency which allows jobs to run on an HPC system. The owners of the resource (cluster/supercomputer) determine how they want the system to be used (often via an allocations committee) over a particular timeframe, often a month, quarter, or year. To enforce their decisions, they distribute allocations to various projects via various accounts and assign each account an account manager. These allocations can be for use particular machines or globally usable. They can also have activation and expiration dates associated with them. All transaction information is typically stored in a database or directory server allowing extensive statistical and allocation tracking.

Each account manager determines how the allocations are made available to individual users within his project. Allocation manager managers such as PNNL's QBank allow the account manager to dedicate portions of the overall allocation to individual users, specify some of allocations as 'shared' by all users, and hold some of the allocations in reserve for later use.

When using an allocations manager each job must be associated with an account. To accomplish this with minimal user impact, the allocation manager could be set up to handle default accounts on a per user basis. However, as is often the case, some users may be active on more than one project and thus have access to more than one account. In these situations, a mechanism, such as a job command file keyword, should be provided to allow a user to specify which account should be associated with the job.

The amount of each job's allocation 'charge' is directly associated with the amount of resources used (i.e. processors) by that job and the amount of time it was used for. Optionally, the allocation manager can also be configured to charge accounts varying amounts based on the QOS desired by the job, the type of compute resources used, and/or the time when the resources were used (both in terms of time of day and day of week).

The allocations manager interface provides near real-time allocations management, giving a great deal of flexibility and control over how available compute resources are used over the medium and long term and works hand in hand with other job management features such as Maui's throttling policies and fairshare mechanism.

Configuring Maui

Maui interfaces with the allocations manager if the parameter BANKTYPE is specified. Maui currently interfaces to QBank, and RES, and can also dump allocation manager interface interaction to a flat file for post processing using the type 'FILE'. Depending on the allocation manager type selected, it may also be necessary to specify how to contact the allocation manager using the parameters BANKSERVER and BANKPORT. When an allocations bank is enabled in this way, Maui will check with the bank before starting any job. For allocation tracking to work, however, each job must specify an account to charge or the bank must be set up to handle default accounts on a per user basis.

Under this configuration, when Maui decides to start a job, it contacts the bank and requests an allocation reservation, or lien be placed on the associated account. This allocation reservation is equivalent to the total amount of allocation which could be consumed by the job (based on the job's wallclock limit) and is used to prevent the possibility of allocation oversubscription. Maui then starts the job. When the job completes, Maui debits the amount of allocation actually consumed by the job from the job's account and then releases the allocation reservation or lien.

These steps transpire 'under the covers' and should be undetectable by outside users. Only when an account has insufficient allocations to run a requested job will the presence of the allocation bank be noticed. If desired, an account may be specified which is to be used when a job's primary account is out of allocations. This account, specified using the parameter BANKFALLBACKACCOUNT is often associated with a low QOS privilege set and priority and often is configured to only run when no other jobs are present.

Reservations can also be configured to be chargeable. One of the big hesitations have 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 configuration a reservation to be chargeable, sites can charge every idle cycle of the reservation to a particular project. When the reservation is in use, the consumed resources will be associated with the account of the job using the resources. When the resources are idle, the resources will be charged to the reservation's charge account. In the case of standing reservations, this account is specified using the parameter SRCHARGEACCOUNT. In the case of administrative reservations, this account is specified via a command line flag to the setres command.

Maui will only interface to the allocations bank when running in 'NORMAL' mode. However, this behavior can be overridden by setting the environment variable 'MAUIBANKTEST' to any value. With this variable set, Maui will attempt to interface to the bank in both SIMULATION and TEST mode.

The allocation manager interface allows you to charge accounts in a number of different ways. Some sites may wish to charge for all jobs run through a system regardless of whether or not the job completed successfully. Sites may also want to charge based on differing usage metrics, such as walltime dedicated or processors actually utilized. Maui supports the following charge policies specified via the parameter BANKCHARGEPOLICY:

DEBITALLWC (charge for all jobs regardless of job completion state using processor weighted wallclock time dedicated as the usage metric)

DEBITSUCCESSFULWC (charge only for jobs which successfully complete using processor weighted wallclock time dedicated as the usage metric)

DEBITSUCCESSFULCPU (charge only for jobs which successfully complete using CPU time as the usage metric)

DEBITSUCCESSFULPE (charge only for jobs which successfully complete using PE weighted wallclock time dedicated as the usage metric)

NOTE: On systems where job wallclock limits are specified, jobs which exceed their wallclock limits and are subsequently cancelled by the scheduler or resource manager will be considered as having successfully completed as far as charging is concerned, even though the resource manager may report these jobs as having been 'removed' or 'cancelled'.

See also BANKTIMEOUT and BANKDEFERJOBONFAILURE.

7.0 Controlling Resource Access - Reservations, Partitions, and QoS Facilities

- 7.1 Advance Reservations
- 7.2 Partitions
- 7.3 QoS Facilities

7.1 Advance Reservations

Reservation Overview

An advance reservation is the mechanism by which Maui guarantees the availability of a set of resources at a particular time. Every reservation consists of 3 major components, a list of resources, a timeframe, and an access control list. It is the job of the scheduler to make certain that the access control list is not violated during the reservation's lifetime (i.e., its timeframe) on the resources listed. For example, a reservation may specify that node002 is reserved for user Tom on Friday. The scheduler will thus be 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, QOS support, and meta scheduling.

7.1.1 Reservation Overview

Every reservation consists of 3 major components, a set of *resources*, a *timeframe*, and an *access control list*. Additionally, a reservation may also have a number of optional attributes controlling its behavior and interaction with other aspects of scheduling. All reservation attributes are described below.

7.1.1.1 Resources

Under Maui, 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. The resources may include processors, memory, swap, local disk, etc. For example, a single task may consist of one 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, Maui will examine all feasible resources and locate the needed resources in groups specified by the task description. An example may help clarify this concept:

Reservation A requires 4 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

In attempting to collect the resources needed for the reservation, Maui would examine each node in turn. Maui 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 utilize these resources according to its own task description and needs.

Currently, the resources which can be associated with reservations include

processors, memory, swap, local disk, initiator classes, and any number of arbitrary resources. Arbitrary resources may include peripherals such as tape drives, software licenses, or any other site specific resource.

7.1.1.2 TimeFrame

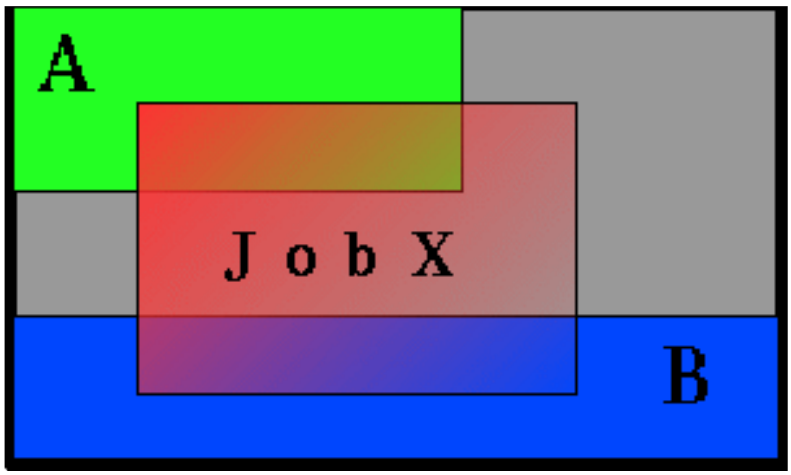
Associated with each reservation is a timeframe. This specifies when the resources will be reserved or dedicated to jobs which meet the reservation's ACL. The timeframe simply consists of a start time and an end time. When configuring a reservation, this information may be specified as a start time together with either an end time or a duration.

7.1.1.3 Access Control List

A reservation's access control list specifies which jobs can use a reservation. Only jobs which meet one or more of a reservation's access criteria are allowed to use the reserved resources during the reservation timeframe. Currently, the reservation access criteria include the following: users, groups, accounts, classes, QOS, and job duration.

7.1.1.4 Job to Reservation Mapping

While a reservation's ACL will allow particular jobs to utilize reserved resources, it does not force any job to utilize these resources. With each job, Maui attempts to locate the best possible combination of available resources whether these are reserved or unreserved. For example, in the figure below, 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.



Although by default, reservations make resources available to jobs which meet particular criteria, Maui 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 can be enabled administratively via a QoS flag. For example, assume two reservations were created as shown below.

```
> setres -g staff -d 8:00:00 'node[1-4]'

reservation 'staff.1' created on 4 nodes

> setres -u john tasks==2

reservation 'john.1' created on two nodes
```

If the user `john`, who happened to also be a member of the group `staff`, wanted to force his job to run within a particular reservation, he could do so using the **FLAGS** resource manager extension. Specifically, in the case of a PBS job, the following submission would force the job to run within the `staff.1` reservation.

```
> qsub -l nodes=1,walltime=1:00:00 -W
x=FLAGS:ADVRES:staff.1 testjob.cmd
```

Note that for this to work, PBS will need to have resource manager extensions enabled as described in the PBS Resource Manager Extension Overview. If the user simply wants the job to run on reserved resources but does not care which, he could submit the job with

```
>qsub -l nodes=1,walltime=1:00:00 -W x=FLAGS:ADVRES
testjob.cmd
```

To enable job to reservation mapping via QoS, the QoS flag '**USERRESERVED**' should be set in a similar manner.

7.1.1.5 Reservation Specification

There are two main types of reservations which 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 setres command. These reservations provide an integrated mechanism to allow graceful management of unexpected system maintenance, temporary projects, and time critical demonstrations. This command allows an administrator to select a particular set of resources or just specify the quantity of resources needed. For example an administrator 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 of use when there is a recurring need for a particular type of resource distribution. For example, a site could use a standing reservation to reserve a subset of its compute resources for quick turnaround jobs during business hours on Monday thru Friday. Standing reservations are created and configured by specifying

parameters in the maui.cfg file. The Standing Reservation Overview provides more information about configuring and using these reservations.

7.1.1.6 Reservation Behavior

As mentioned above, a given reservation may have one or more access criteria. A job can utilize 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 utilize the given node if it meets at least access criteria of each active reservation on the node.

7.1.1.7 Other Reservation Attributes

Charge Account - Allows a reservation to charge for resources which are dedicated to the reservation but not used by any job.

See also:

N/A

7.1.2 Administrative Reservations

Administrative reservations behave much like standing reservations but are generally created to address non-periodic, 'one time' issues. All admin reservations are created using the setres command and are persistent until they expire or are removed using the releaseres command.

See also: Reservation Overview, Backfill

7.1.3 Standing Reservations

Standing reservations build upon the capabilities of advance reservations to enable a site 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 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 a site 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 PM. See the Reservation Overview for more information about the use of reservations. The Managing Reservations section 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
Controlling Access to Resources
Controlling Turnaround

see the following parameters for more information: SRNAME SRRESOURCES SRDAYS SRFLAGS SRSTARTTIME SRENDTIME SRWSTARTTIME SRWENDTIME SRDEPTH SRTASKCOUNT SRHOSTLIST SRTPN SRUSERLIST SRGROUPLIST SRACCOUNTLIST SRQOSLIST SRCLASSLIST SRMAXTIME SRTIMELOGIC SRPARTITION SRACCESS

7.1.4 Reservation Policies

In addition to standing and administrative reservations, Maui 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 indefinate period. In the case of backfill, small jobs may continue to be run on available resources as they become available while a large job sits in the queue never able to find enough nodes available simultaneously to run on. To avoid such situations, priority reservations are created for high priority jobs which 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.

By default, only the highest priority job will receive a priority reservation. However, this behavior is configurable via the RESERVATIONDEPTH policy. Maui's default behavior of only reserving the highest priority job allows backfill to be used in a form known as *liberal* backfill. This 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 may 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 which cannot run is given a reservation. Most sites prefer the liberal backfill approach associated with the default **RESERVATIONDEPTH** of 1 or select a slightly higher value. It is important to note that to prevent starvation in conjunction with reservations, monotonically increasing priority factors such as queuetime or job xfactor should be enabled. See the Prioritization Overview for more information on priority factors.

Another important consequence of backfill and reservation depth is its affect on job priority. In Maui, all jobs are prioritized. Backfill allows jobs to be run out of order and thus, to some extent, job priority to be ignored. This effect, known as 'priority dilution' can cause many site policies implemented via Maui prioritization policies to be ineffective. Setting the **RESERVATIONDEPTH** parameter to a higher value will give 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%.

In addition to **RESERVATIONDEPTH**, sites also have the ability to control how reservations are maintained. Maui's dynamic job prioritization allows sites to prioritize jobs so that their priority order can change over time. 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 allows a site to determine what how existing reservations should be handled when new reservations are made. The value **HIGHEST** will

cause that all jobs which have ever received a priority reservation will maintain that reservation until they run even if other jobs later bypass them in priority value. The value **CURRENTHIGHEST** will cause that only the current top <RESERVATIONDEPTH> priority jobs will receive reservations. If a job had a reservation but has been bypassed in priority by another job so that it no longer qualifies as being amongst the top <RESERVATIONDEPTH> jobs, it will lose its reservation. Finally, the value **NEVER** indicates that no priority reservations will be made.

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 QoS's. For example, the following configuration will create two reservation depth *groupings*:

```
RESERVATIONDEPTH[0]   8
RESERVATIONQOSLIST[0] highprio interactive debug

RESERVATIONDEPTH[1]   2
RESERVATIONQOSLIST[1] batch
```

This example will cause that the top 8 jobs belonging to the aggregate group of `highprio`, `interactive`, and `debug` QoS jobs will receive priority reservations. Additionally, the top 2 `batch` QoS jobs will also receive priority reservations. Use of this feature allows sites to maintain high throughput for important jobs by guaranteeing the a significant proportion of these jobs are making progress towards starting through use of the priority reservation.

A final reservation policy is in place to handle 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, Maui 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. Thus, when the original job reservation is re-established, its original resource may be unavailable and the resulting new reservation may be delayed several hours from the earlier start time. The parameter RESERVATIONRETYTIME allows a site that is experiencing frequent resource manager race conditions and/or corruption situations to tell Maui to hold on to the reserved resource for a period of time in an attempt to allow the resource manager to correct its state.

See also: Reservation Overview, Backfill

7.1.5 Configuring and Managing Reservations

All reservations, whether they be administrative or standing, possess many similar traits.

7.1.5.1 Reservation Attributes

All reservations possess a timeframe of activity, an access control list, and a list of resources to be reserved. Additionally, reservations may also possess a number of extension flags which modify the behavior of the reservation.

7.1.5.1.1 Start/End Time

All reservations possess a start and an end time which define the reservation's 'active' time. During this active time, the resources within the reservation may only be used as specified by the reservation ACL. This active time may 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.

7.1.5.1.2 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. This is handled by way of an access control list, or ACL.

7.1.5.1.3 Resources

When specifying which resources to reserve, the administrator has a number of options. These options allow control over how many resources are reserved and where they are reserved at. The following reservation attributes allow the administrator to define resources

Task Description

A key concept of reservations is the idea of a *task*. Maui 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, Maui 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 Maui must collect resources for use in a standing reservation, however, they do not constrain the way in which Maui 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 allocated 6 tasks with 2 processors and 512 MB of memory each, it could support job Y which requires 10 tasks of 1 processor and 128 MB of memory or job Z which requires 2 tasks of 4 processors and 1 GM of memory each. The task constraints used to acquire a reservation's resources are completely transparent to a job requesting use of these resources.

Task Count

Using the task description, the task count attribute defines how many tasks must be collected to satisfy the reservation request. To create a reservation, a task count and/or a hostlist must be specified

Hostlist

A hostlist constrains the set of resource which are available to a reservation. If no task count is specified, the reservation will attempt to reserve one task on each of the listed resources. If a task count is specified which requests fewer resources than listed in the hostlist, Maui will reserve only <TASKCOUNT> tasks from the hostlist nodes. If a taskcount is specified which requests more resources than listed in the hostlist, Maui will reserve the hostlist nodes first and then seek additional resources outside of this list.

7.1.5.1.4 Flags

Reservation flags allow specification of special reservation attributes or behaviors. The following flags are supported:

Flag Name	Description
BYNAME	reservation will only allow access to jobs which meet reservation ACL's **and** explicitly request the resources of this reservation using the job ADVRES flag
PREEMPTEE	N/A
BESTEFFORT	N/A

7.1.5.2 Configuring Standing Reservations

Standing reservations allow resources to be dedicated for particular uses 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. All standing reservation attributes are specified via parameters. An overview of standing reservation capabilities is included below followed by a series of examples.

7.1.5.2.1 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 a standing reservation recurs 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 SRDEPTH parameter.

For example, the following configuration will create a standing reservation for 6 processors and 3 GB of memory for use by the interactive class during business hours.

```
---
SRNAME[0]        interactive
SRTASKCOUNT[0]   6
SRRESOURCES[0]   PROCS=1;MEM=512
SRPERIOD[0]      DAY
SRDAYS[0]        MON TUE WED THU FRI
SRSTARTTIME[0]   9:00:00
SRENDTIME[0]     17:00:00
SRCLASSLIST[0]   interactive
---
```

In Maui 3.2.0 or later, this could be accomplished using the SRCFG parameter as in the example below:

```
---
SRCFG[interactive] STARTTIME=9:00:00
ENDTIME=17:00:00
SRCFG[interactive] PERIOD=DAY
DAYS=MON,TUE,WED,THU,FRI
SRCFG[interactive] TASKCOUNT=6
RESOURCES=PROCS:1;MEM:512
SRCFG[interactive] CLASSLIST=interactive
---
```

Let's examine the new parameters one at a time. SRNAME simply gives the standing reservation a name for reference by Maui commands. It is not required but makes administration a bit easier. SRTASKCOUNT allocated in units called 'tasks' where a task is a

collection of resources which must be allocated together on a single node. The next parameter, SRRESOURCES, indicates what resources must be included in each task. In this case, Maui must locate and reserve 1 processor and 512 MB of memory together on the same node for each task requested. SRPERIOD states that this reservation is periodic on a daily basis with the actual days of the week which the standing reservation should be enabled specified using SRDAYS. The time of day during which the requested tasks are to be reserved are specified using SRSTARTTIME and SRENDTIME. Finally, the SRCLASSLIST parameter is used to indicate that jobs requesting the class `interactive` should be allowed to use this reservation.

7.1.5.2.2 Specifying Reservation Resources

This is a lot of new information to digest. Not all of the parameters are needed in all cases. For example, by default, **SRRESOURCES** 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 for all uniprocessor systems, this default behavior is probably easiest to work with. However, when SMP systems are thrown into the mix, **SRRESOURCES** provides a powerful means of specifying an exact, multi-dimensional resource set.

An examination of the parameters documentation will show that the default value of **SRPERIOD** is **DAYS**. Thus, specifying this parameter in the example above was unnecessary. It was used only to introduce this parameter and indicate that other options exist beyond daily standing reservations.

Hopefully, the next few examples will further clarify the use of standing reservations while expanding on some of the intricacies surrounding their use. Note that the above example did not specify where the needed six tasks were to be located. If this information is not specified, Maui will attempt to locate the needed resources anywhere it can find them. The reservation will essentially *float* to nodes where the needed resources can be found.

Let's assume you actually wanted to constrain this reservation to a particular set of resources. In this case, the parameter SRHOSTLIST can be used to specify which nodes can be used for the reservation. The following example will do this.

```
-----
SRNAME[0]        interactive
```

```
SRHOSTLIST[0]    node003  node004  node005
node011  node012  node052
SRTASKCOUNT[0]   6
SRRESOURCES[0]   PROCS=1;MEM=512
SRDAYS[0]        MON TUE WED THU FRI
SRSTARTTIME[0]   9:00:00
SRENDTIME[0]     17:00:00
SRCLASSLIST[0]   interactive
-----
```

The example is now a bit more complex. Note that we added a non-contiguous list of nodes where the standing reservation can locate the needed resources using the **SRHOSTLIST** parameter. It is important to note that the fact that there is a one to one mapping between **SRTASKCOUNT** and the hosts in **SRHOSTLIST** does not necessarily mean that Maui will place one task on each host. If, for example, node011 and node012 were 2 processor SMP nodes with 1 GB of memory, Maui could locate 2 tasks on each of these nodes leaving only two more tasks to be placed. (Maui will place tasks on nodes 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 **SRTASKCOUNT**, Maui will simply select the needed resources within the set of nodes listed.

If **SRHOSTLIST** is specified but **SRTASKCOUNT** is not, Maui will pack as many tasks as it can onto ALL of the listed nodes. For example,

```
-----
SRNAME[1]        debug
SRHOSTLIST[1]    node001 node002 node003
node004
SRUSERLIST[1]    helpdesk
SRGROUPLIST[1]   operations sysadmin
SRPERIOD[1]      INFINITY
-----
```

This standing reservation appears much simpler. Since **SRRESOURCES** is not specified, it will allocate all processors on each of the nodes listed in **SRHOSTLIST**. Since a start and end time are not specified, the reservation will be in force all day long. Since **SRDAYS** is not specified, the reservation will be enabled every day of the week.

Ok, here come a couple of curve balls. First, note that standing reservation 1, 'debug', has two access parameters set,

SRUSERLIST, and SRGROUPLIST. Reservations can be accessed by any one of the access lists specified. In this case, either user 'helpdesk' or any member of the groups 'operations' or 'sysadmin' can use these resources. While access is granted to the logical 'OR' of access lists specified within a standing reservation, access is only granted to the logical *AND* of access lists across different standing reservations. Come again? Compare standing reservations `interactive` and `debug` in the examples above. Note that they both can allocate nodes `node003` and `node004`. Assume that node003 had both of these reservations in place simultaneously and a job attempted to access this node during business hours when standing reservation `interactive` was active. The job could only use the *doubly* reserved resources if it requested the run class `interactive`*AND* it met the constraints of reservation `debug` (i.e., was submitted by user '`helpdesk`' or by a member of the group `operations` or `sysadmin`).

7.1.5.2.3 Reservation Stacking

To make things just a little more confusing, Maui will not *stack* reservations unless it has to. If adequate resources exist, it can allocate reserved resources *side by side* in a single SMP node rather than on top of each other. Take the case of a 16 processor SMP node with two 8 processor standing reservations. Eight of the processors on this node will be allocated to the first reservation, and eight to the next. Any configuration is possible. The 16 processor nodes 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 sites choose to do it to enforce elaborate policies. There is no problem with doing so so long as you can keep things straight. It really is not too difficult a concept, just takes a little getting used to. See the 'Reservation Overview' section for a more detailed description of reservation use and constraints. I am working on extending reservation ACL's to allow cleaner arbitrary ACL list support but there are some significant scheduling performance hits associated with completely general ACL support.

Now for another example. As mentioned earlier, by default Maui enforces standing reservations by creating a number of reservations where the number created is controlled by the SRDEPTH parameter. When Maui starts up, and again each night at midnight, Maui updates its periodic, *non-floating* standing reservations. By default, **SRDEPTH** is set to 2, meaning when Maui starts up, it will create two 24 hour reservations covering two days worth of time,

(i.e. a reservation for today and one for tomorrow.) At midnight, today's reservation will be expired and removed, tomorrow's reservation will become today's and Maui will create a new reservation for the next day. Maui continues creating reservations in the future as time continues its incessant march forward. Everything's great, resources are always reserved as needed when today rolls around. Then what's this **SRDEPTH** parameter for? This parameter remedies a situation which might occur when a job is submitted and cannot run immediately because the system is completely backlogged with jobs. In such a case, available resources may not exist for two days out and Maui will reserve them for this job. When midnight arrives, Maui attempts to *roll* its standing reservations but here a problem arises! This job has now allocated the resources needed for the standing reservation two days out! Maui cannot reserve the resources for the standing reservation because they are already claimed by the job. The standing reservation reserves what it can but it is now smaller than it should be or possibly even empty.

If a standing reservation is smaller than it should be, Maui will attempt to add resources every iteration until it is fully populated. However, in the case of this job, it is not going to let go of the resources it has and the standing reservation is out of luck. The **SRDEPTH** parameter allows a site to create standing reservations deep into the future allowing them to claim the resources first and preventing this problem. If *partial* standing reservations are detected on a system, it may be an indication that the **SRDEPTH** parameter should be increased.

In the example above, the **SRPERIOD** parameter 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.

One final example. It was claimed earlier that access lists within a reservation are OR'd together to determine reservation access. However, this rule has one notable exception triggered by use of the parameter SRMAXTIME. This parameter controls the length of time a job can use the resources in a standing reservation. This access mechanism can be AND'd or OR'd to the cumulative set of all other access lists as specified by the SRTIMELOGIC parameter. Consider the following example configuration:

```
---
SRNAME[0]         shortpool
SRTASKCOUNT[0]    32
```

```
SRPERIOD[0]       WEEK
SRWSTARTTIME[0]   1:08:00:00
SRWENDTIME[0]     5:17:00:00
SRFEATURES[0]     largememory
SRMAXTIME[0]      1:00:00
SRTIMELOGIC[0]    AND
SRQOSLIST[0]      high low special-
SRACCOUNTLIST[0] !projectX !projectY
-----
```

The term *final example* probably made it sound like we were just about finished, didn't it? So why are there 1, 2, 3, ... 7 new parameters in this example? Our apologies, we figure only the die hards are reading at this point! In a nutshell, this specification asks for 32 tasks which translate to 32 nodes. **SRPERIOD** states that this reservation is periodic on a weekly basis while the parameters SRWSTARTTIME and SRWENDTIME specify the week offsets when this reservation is to start and end. In this case, the reservation starts on Monday at 8:00 AM and runs until Friday at 5:00 PM. The reservation is enforced as a series of weekly reservations which only cover the specified timeframe. The SRFEATURES parameter indicates that each of these nodes must have the node feature `largememory` configured.

As described above, **SRMAXTIME** indicates that jobs using this reservation can only use it for one hour. What does this mean? It 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, so does a four hour job that starts on Monday at 5:00 AM or a 12 hour job which starts on Friday at 4:00 PM. Also, note the **SRTIMELOGIC** setting. It is set to `AND`. This means that jobs must not only meet the **SRMAXTIME** 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 utilize the access specified via SRQOSLIST or SRACCOUNTLIST, i.e., it is assigned a QOS of `high`, `low`, or `special`, or the submitter of the job has an account which satisfies the `!projectX` and `!projectY` criteria (More on this below). **NOTE**: See the QOS Overview for more info about QOS configuration and usage

7.1.5.2.4 Affinity

One aspect of reservations that has not yet been discussed is something called reservation affinity. By default, jobs *gravitate* towards reservations in a behavior known as *positive affinity*. This

allows jobs to run on the most constrained resources leaving other, unreserved resources free for use by other jobs which may not be able to access the reserved resources. Normally this is a desired behavior. However, sometimes, it is desirable to reserve resources for use as a *last resort*, ie use the reserved resources only when there are no other resources available. This 'last resort' behavior is known as 'negative affinity'. Note the '-' (dash or negative sign) following the '`special`' in the **SRQOSLIST** values above. This is not a typo, rather it indicates that QOS '`special`' should be granted access to this reservation but should be assigned negative affinity. Thus, the **SRQOSLIST** parameter 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 '=' character, ie '`SRQOSLIST[0] normal= high debug= low-`'.

In addition to affinity, ACL's may also be of different types. Note the **SRACCOUNTLIST** values in the previous example. They 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 (ie '!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`' will be denied access to the reservation even though the job QOS matches the QOS ACL. **Note that the ability to specify 'NOT' ACLs is only enabled in Maui 3.0.7 and higher.**

7.1.5.2.5 Resource Allocation Behavior

As mentioned above, standing reservations can operate in one of two modes, floating, or non-floating (essentially node-locked). A floating reservation is created when a **SRTASKCOUNT** is specified and **SRHOSTLIST** is either not specified or specified with more resources than are needed to fulfill the **SRTASKCOUNT** requirement. If a reservation is non-floating, Maui will allocate all resources specified by the **SRHOSTLIST** parameter regardless of node state, job load, or even the presence of other standing reservations. Maui interprets the request for a non-floating reservation as stating, 'I want a reservation on these exact nodes, no matter what!'

If a reservation is configured to be floating, Maui takes a more relaxed stand, searching through all possible nodes to find resources meeting standing reservation constraints. Only Idle, Running, or Busy node will be considered and further, only considered if no reservation conflict is detected. The parameter SRACCESS can be used to modify this behavior slightly and allow the reservation to allocate resources even if reservation conflicts exist.

Other standing reservation parameters not covered here include SRPARTITION and SRCHARGEACCOUNT. These parameters are described in some detail in the Maui parameters documentation.

7.1.5.3 Configuring Administrative Reservations

A default reservation, with no ACL, is termed 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.

(Under construction)

See Also:

N/A

7.2 Partitions

Partitions are a logical construct which divide available resources. By default, a given job may only utilize resources within a single partition and any resource (i.e., compute node) may only be associated with a single partition. In general, partitions are organized along physical or political boundaries. For example, a cluster may consist of 256 nodes containing four 64 port switches. This cluster may receive excellent interprocess communication speeds for parallel job tasks located within the same switch but sub-stellar performance for tasks which 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 Maui, the standing reservation facility provides significantly improved flexibility and should be used in the vast majority of cases where partitions are required under other resource management systems. Standing reservations provide time flexibility, improved access control features, and more extended resource specification options. Also, another Maui 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 earlier, many of Maui's scheduling policies may be specified on a per partition basis allowing each department to control the scheduling goals within their partition.

The partition associated with each node must be specified as indicated in the Node Location section. With this done, partition access lists may be specified on a per job or per QOS basis to constrain which resources a job may have access to (See the QOS Overview for more information). By default, QOS's and jobs allow global partition access.

If no partition is specified, Maui creates a single partition named '**DEFAULT**' into which all resources are placed. In addition to the DEFAULT partition, a pseudo-partition named '**[ALL]**' is created which contains the aggregate resources of all partitions. NOTE: While DEFAULT is a real partition containing all resources not explicitly assigned to another partition, the [ALL] partition is only a convenience construct and is not a real partition; thus it cannot be requested by jobs or included in configuration ACL's.

7.2.1 Defining Partitions

7.2.1 Defining Partitions

Node to partition mappings are established using the NODECFG parameter in Maui 3.0.7 and higher as shown in the example below.

```
---
NODECFG[node001]   PARTITION=astronomy
NODECFG[node002]   PARTITION=astronomy
...
NODECFG[node049]   PARTITION=math
...
---
```

In earlier versions of Maui, node to partition mappings were handled in the machine config file (machine.cfg) using the PARTITION keyword as in the example below.

```
---
node001   PARTITION=astronomy
node002   PARTITION=astronomy
...
node049   PARTITION=math
...
---
```

However, if using partitions, it is HIGHLY recommended that Maui 3.0.7 or higher be used.

7.2.2 Managing Partition Access

Determining who can use which partition is specified using the ***CFG** parameters (USERCFG, GROUPCFG, ACCOUNTCFG, QOSCFG, CLASSCFG, and SYSTEMCFG). These parameters allow both a partition access list and default partition to be selected on a credential or system wide basis using the PLIST and PDEF keywords. By default, the access associated with any given job is the 'logical or' of all partition access lists assigned to the job's credentials. Assume a site with two partitions, general, and test. The site management would like 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 mgmt 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 example configuration below will enable the needed user

and group access and defaults for this site.

```
USERCFG[DEFAULT] PLIST=general
USERCFG[steve]   PLIST=general:test PDEF=test
GROUPCFG[staff]  PLIST=general:test PDEF=general
GROUPCFG[mgmt]   PLIST=general:test PDEF=general
```

Note that the DEFAULT user has no default partition specified. If only a single partition is provided in the access list, it will be selected as the default partition.

In Maui 3.0.6 and earlier, partition access would be controlled using the following stanza in the fairshare config file (fs.cfg)

```
USER:DEFAULT PLIST=general
USER:steve   PLIST=general:test PDEF=test
GROUP:staff  PLIST=general:test PDEF=general
GROUP:mgmt   PLIST=general:test PDEF=general
```

7.2.3 Requesting Partitions

Users may 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 PBS 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 '#PBS -W x=PARTITION:test' to the command file. See the resource manager extension overview for more information on configuring and utilizing resource manager extensions.

7.2.4 Miscellaneous Partition Issues

Special jobs may be allowed to span the resources of multiple partitions if desired by associating the job with a QOS which has the flag '**SPAN**' set. (See the QOSCFG parameter)

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.

See Also:

N/A

7.3 Quality of Service (QoS) Facilities

7.3.1 QoS Overview

The QOS facility allows a site to give special treatment to various classes of jobs, users, groups, etc. Each QOS object can be thought of 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 account which can access these privileges.

Sites can configure various QOS's 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 QOS's. A given job will have a default QOS and may have access to several additional QOS's. When the job is submitted, the submittor may request a specific QOS (see the User's Manual for information on specifying job QOS for the resource manager of interest) or just allow the default QOS to be used. Once a job is submitted, a user may adjust the QOS of his job's at any time using the 'setqos' command. The **setqos** command will only allow the user to modify the QOS of his jobs and only change the QOS to a QOS that this user has access to. Maui administrators may change the QOS of any job to any value.

Jobs are currently granted access to a QOS privileges by configuring QDEF (QOS Default) or QLIST (QOS Access List) settings in the fs.cfg file. A job may access a particular QOS if that QOS is listed in the system default configuration QDEF or QLIST, or if the QOS is specified in the QDEF or QLIST of a user, group, account, or class associated with that job.

The 'diagnose -Q' command can be used to obtain information about the current QOS configuration.

7.3.2 QoS Enabled Privileges

The privileges enabled via QoS settings may be broken into one of the following categories

Special Prioritization
Service Access and Constraints
Override Policies and Policy Exemptions

All privileges are managed via the QOSCFG parameter.

7.3.2.1 Special Prioritization

Attribute Name	Description
FSTARGET	
PRIORITY	Assign priority to all jobs requesting particular QoS
QTTARGET	
QTWEIGHT	
XFTARGET	
XFWEIGHT	

Example:

```
---
QOSCFG[geo]  PRIORITY=10000
---
```

7.3.2.2 Service Access and Constraints

The QoS facility can ne used to enable special service and/or disable default services. All services are enabled/disabled by setting the QoS **FLAG** attribute.

Flag Name	Description
DEDICATED	jobs should not share compute resources with any other job. These jobs will only run on nodes which are idle and will not allow other jobs to use resources on allocated nodes even if additional resources are available.
NOBF	job cannot be considered for backfilled
NORESERVATION	job should never reserve resources regardless of priority
PREEMPTEE	job may be preempted by higher priority PREEMPTOR jobs
PREEMPTOR	job may preempt lower priority PREEMPTEE jobs
RESERVEALWAYS	job should create resource reservation regardless of job priority

RESTARTPREEMPT	jobs can preempt restartable jobs by essentially requeueing them if this allows the QOS job to start earlier
USERESERVED[:<RESID>]	job may only utilize resources within accessible reservations. If <RESID> is specified, job may only utilize resources within the specified reservation.

Example:

```
QOSCFG[hiprio]    FLAGS=NOBF:PREEMPTEE
```

Example 2:

```
QOSCFG[chem-b]  FLAGS=USERESERVED:chemistry
```

7.3.2.3 Policy Exemptions

Individual QoS's may be assigned override policies which will set new policy limits regardless of user, group, account, or queue limits. Particularly, the following policies may be overridden:

MAXJOB
MAXPROC
MAXNODE

Example:

```
QOSCFG[staff]  MAXJOB=48
```

In addition to overriding policies, QoS's may also be used to allow particular jobs to ignore policies by setting the QoS **FLAG** attribute

QOS Flags

IGNJOBPERUSER
IGNPROCPERUSER
IGNPSPERUSER
IGNJOBQUEUEDPERUSER
IGNJOBPERGROUP
IGNPROCPERGROUP
IGNPSPERGROUP

IGNJOBQUEUEDPERGROUP
IGNJOBPERACCOUNT
IGNPROCPERACCOUNT
IGNPSPERACCOUNT
IGNJOBQUEUEDPERACCOUNT
IGNSYSMAXPROC
IGNSYSMAXTIME
IGNSYSMAXPS

IGNSRMAXTIME

jobs should ignore standing reservation MAXTIME constraints

IGNUSER

jobs should ignore all user throttling policies

IGNGROUP

jobs should ignore all group throttling policies

IGNACCOUNT

jobs should ignore all account throttling policies

IGNSYSTEM

jobs should ignore all system throttling policies

IGNALL

jobs should ignore all user, group, and account throttling policies

Example

```
QOSCFG[express]  FLAGS=IGNSYSTEM
```

7.3.3 Managing QoS Access

Managing which jobs can access which privileges is handled via the QOSCFG parameter. Specifically, this parameter allows the specification of a access control list based on a job's user, group, account, and queue credentials. To enable QoS access, the **QLIST** and/or **QDEF** attributes of the appropriate user, group, account, or queue should be specified using the parameters USERCFG, GROUPCFG, ACCOUNTCFG, and CLASSCFG respectively.

Example:

```
USERCFG[john]     QDEF=geo  QLIST=geo,chem,staff
GROUPCFG[systems] QDEF=development
CLASSCFG[batch]   QDEF=normal
```

See also:

N/A

8.0 Optimizing Scheduling Behavior - Backfill, Node Sets, and Preemption

---

8.1 Optimization Overview

Under Construction

8.2 Backfill

8.2.1 Backfill Overview

Backfill is a scheduling optimization which allows a scheduler to make better use of available resources by running jobs out of order. When Maui schedules, it prioritizes the jobs in the queue according to a number of factors and then orders the jobs into a 'highest priority first' sorted list. It starts the jobs one by one stepping through the priority list until it reaches a job which it cannot start. Because all jobs and reservations possess a start time and a wallclock limit, Maui can determine the completion time of all jobs in the queue. Consequently, Maui 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 Maui 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, Maui, 'protects' the highest priority job's start time by creating a job reservation to reserve the needed resources at the appropriate time. Maui then can any job which not not interfere with this reservation.

Backfill offers significant scheduler performance improvement. In a typical large system, enabling backfill will increase system utilization by around 20% and improve 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 only moderate to no improvement for the larger, long ones.

The question arises, is backfill a purely good feature. Doesn't there have to be a trade-off some where? Doesn't there have to be a dark side? Well, there are a few drawbacks to using backfill but they are fairly minor. First of all, because backfill locates jobs to run scattered throughout the idle job queue, it tends to diminish the influence of the job prioritization a site has chosen and thus may negate any desired workload steering attempts through this prioritization. Secondly, although the start time of the highest priority job is protected by a reservation, what is to prevent the third priority job from starting early and possibly delaying the start of the second priority job? Ahh, a problem. Actually, one that is easily handled as

will be described later.

The third problem is actually a little more subtle. Consider the following scenario involving the 2 processor cluster shown in figure 1. Job A has a 4 hour wallclock limit and requires 1 processor. It started 1 hour ago and will reach its wallclock limit in 3 more hours. Job B is the highest priority idle job and requires 2 processors for 1 hour. Job C is the next highest priority job and requires 1 processor for 2 hours. Maui examines the jobs and correctly determines that job A must finish in 2 hours and thus, the earliest job B can start is in 2 hours. Maui also determines that job C can start and finish in less than this amount of time. Consequently, Maui starts job C on the idle processor. One hour later, job A completes early. Apparently, the user overestimated the amount of time his job 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. Maui re-evaluates job B's reservation and determines that it can be slid forward an hour. At time 3, job B starts.

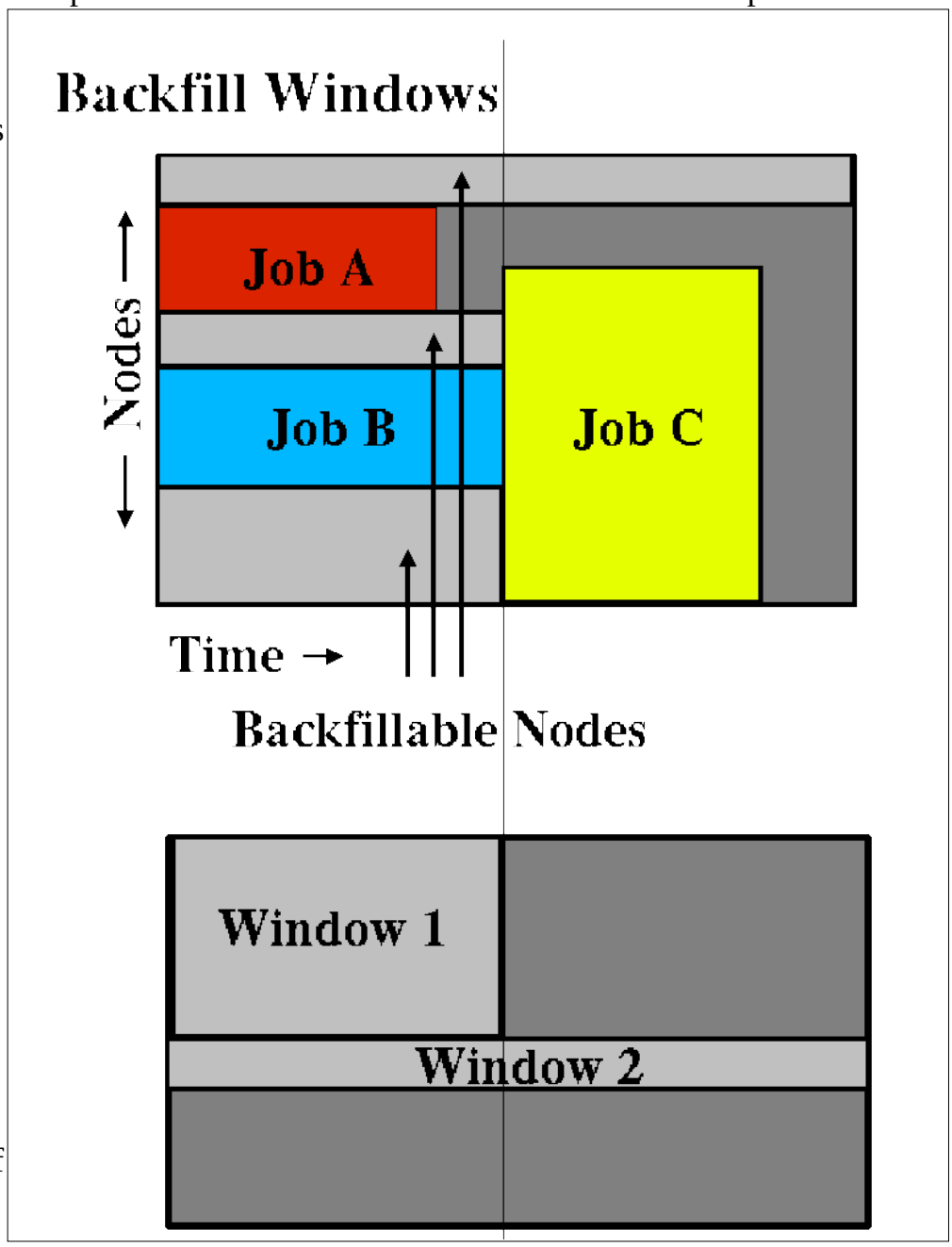
Ok, now the post-game show. Job A is happy because it ran to completion. Job C is happy because it got to start immediately. Job B is sort of happy because it got to run 1 hour sooner than it originally was told it could. However, if backfill was not enabled, job B would have been able to run 2 hours earlier. Not a big deal, usually. However, the scenario described above actually occurs fairly frequently. This is because the user estimates for how long their jobs will take is generally very bad. 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 40%. 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.

So, is backfill worth it? The short answer is absolutely. The longer answer is a.....b.....s.....o.....l.....u.....t.....e.....l.....y. Although there do exist some minor drawbacks with backfill, its net performance impact on a site's workload is very positive. Its like the phrase 'a rising tide lifts a ships'. Although a few of the highest priority jobs may get minorly and temporarily delayed, they probably got to their position as highest priority as soon as they did because jobs in front of them got to run earlier due to backfill. Studies have shown that only a very small fraction 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. Regarding the other problems described, 'don't vorry, ve have vays of handling dem.'

8.2.2 Backfill Algorithm

The algorithm behind Maui backfill scheduling is mostly straightforward although there are a number of issues and parameters of which you should be aware. First of all, Maui makes two backfill scheduling passes. For each pass, Maui selects a list of jobs which are eligible for backfill. On the first pass, only those jobs which meet the constraints of the 'soft' fairness throttling policies are considered and scheduled. The second pass expands this list of jobs to include those which meet the 'hard' (less constrained) fairness throttling policies.

The second important concept regarding Maui 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 grey boxes represent currently idle nodes which are eligible for backfill. For this example, lets assume that the space represented covers 8 nodes and a 2 hour timeframe. To determine backfill windows, Maui 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 which are available for only one hour (because some of the nodes are blocked by the reservation for job C). The second window contains only one node but has no time limit because this node is not blocked by the reservation for job C. It is important to note that these backfill windows overlap.



Once the backfill windows have been determined, Maui begins to traverse them. The current behavior is to traverse these windows 'widest window first' (i.e., most nodes to fewest nodes) As each backfill window is evaluated, Maui applies the backfill algorithm specified by the BACKFILLPOLICY parameter, be it **FIRSTFIT**, **BESTFIT**, etc.

Assuming the **BESTFIT** algorithm is applied, the following steps are taken.

1) The list of feasible backfill jobs is filtered, selecting only those which will actually fit in the current backfill window.
2) The 'degree of fit' of each job is determined based on the SCHEDULINGCRITERIA parameter (ie, processors, seconds, processor-seconds, etc) (ie, if processors is selected, the job which requests the most processors will have the

best fit)
3) The job with the best fit is started.
4) While backfill jobs and idle resources remain, repeat step 1.

Other backfill policies behave in a generally similar manner. The parameters documentation can provide further details.

One final important note. By default, Maui reserves only the highest priority job resulting in a very 'liberal' and aggressive backfill. This reservation guarantees that backfilled jobs will not delay the highest priority job, although they may delay the second highest priority job! (Actually, due to wallclock inaccuracies, it is possible the the highest priority job may actually get slightly delayed as well but we won't go into that!) The parameter RESERVATIONDEPTH controls how conservative/liberal the backfill policy is. This parameter controls how deep down the priority queue to make reservations. While increasing this parameter will improve guarantees that priority jobs will not be bypassed, it reduces the freedom of the scheduler to backfill resulting in somewhat lower system utilization. The value of the trade-offs often need to be determined on a site by site basis.

8.2.3 Configuring Backfill

Backfill is enabled in Maui by specifying the BACKFILLPOLICY parameter. By default, backfill is enabled in Maui using the FIRSTFIT algorithm. However, this parameter can also be set to BESTFIT, GREEDY, or NONE. The number of reservations can also be controlled using RESERVATIONDEPTH[<X>] This depth can be distributed across job QOS levels using RESERVATIONQOSLIST[<X>].

See also:

Parameters **BACKFILLDEPTH**and **BACKFILLMETRIC**
Reservation Policy Overview.

8.3 Node Set Overview

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 as well as 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 thus 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 and can be based on node processor speed, memory, network interfaces, or locally defined node attributes. In addition to their use in forcing jobs onto homogeneous nodes, these policies may 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 may run best on a certain range of processor speeds, running slower on slower nodes, while wasting cycles on faster nodes. A job may specify ANYOF:PROCSPEED:450,500,650 to request nodes in the range of 450 to 650 MHz. Alternatively, if a simple procspeed-homogeneous node set is desired, ONEOF:PROCSPEED may be specified. On the other hand, a communication sensitive job may request a network based node set with the configuration ONEOF:NETWORK:via,myrinet,ethernet, in which case Maui will first attempt to locate adequate nodes where all nodes contain via network interfaces. If such a set cannot be found, Maui will look for sets of nodes containing the other specified network interfaces. In highly heterogeneous clusters, the use of node sets have been found to improve 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. In all cases, node sets are a dynamic construct, created on a per job basis and built only of nodes which meet all of the jobs requirements.

As an example, let's assume a large site possessed a Myrinet based interconnect and wished 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 (ie, switchA, switchB, etc) and then use the following system wide node set configuration:

```
----
NODESETPOLICY    ONEOF
NODESETATTRIBUTE FEATURE
```

```
NODESETDELAY     0:00:00
NODESETLIST      switchA switchB switchC switchD
----
```

The NODESETPOLICY parameter tells Maui to allocate nodes within a single attribute set. Setting NODESETATTRIBUTE to **FEATURE** specifies that the node sets are to be constructed along node feature boundaries. The next parameter, NODESETDELAY, indicates that Maui 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 zero basically tells Maui to attempt to use a node set if it is available, but if not, run the job as soon as possible anyway. Finally, the NODESETLIST value of 'switchA switchB...' tells Maui to only use node sets based on the listed feature values. This is necessary since sites will 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.

On occasion, sites may wish to allow a less strict interpretation of nodes sets. In particular, many sites seek to enforce a more liberal **PROCSPEED** based node set policy, where *almost balanced* node allocations are allowed but wildly varying node allocations are not. In such cases, the parameter NODESETTOLERANCE may be used. This parameter allows specification of the percentage difference between the fastest and slowest node which can be within a nodeset using the following calculation:

(Speed.Max - Speed.Min) / Speed.Min <= **NODESETTOLERANCE**

Thus setting **NODESETTOLERANCE** to 0.5 would allow the fastest node in a particular node set to be up to 50% faster than the slowest node in that set. With a 0.5 setting, a job may allocate a mix of 500 and 750 MHz nodes but not a mix of 500 and 900 MHz nodes. Currently, tolerances are only supported when the **NODESETATTRIBUTE** parameter is set to **PROCSPEED**. The MAXBALANCE node allocation algorithm is often used in conjunction with tolerance based node sets.

When resources are available in more than one resource set, the NODESETPRIORITYTYPE parameter allows control over how the 'best' resource set is selected. Legal values for this parameter are described in the table below.

Priority Type	Description	Details
BESTFIT	select the smallest resource set possible	minimizes fragmentation of larger resource sets.
BESTRESOURCE	select the resource set with the 'best' nodes	only supported when `NODESETATTRIBUTE` is set to `PROCSPEED`. Selects the fastest possible nodes for the job.

MINLOSS	select the resource set which will result 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)	Only supported when NODESETATTRIBUTE is set to PROCSPEED and NODESETTOLERANCE is > 0. This algorithm is highly useful in environments with mixed speed compute nodes and a non load-balancing parallel workload.
WORSTFIT	select the largest resource set possible	minimizes the creation of small resource set fragments but fragments larger resource sets.

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.

See also:

N/A.

8.4 Preemption Policies (enabled in Maui 3.0.7 and above)

Many sites possess workloads of varying importance. While it may be critical that some jobs obtain resources immediately, other jobs are less *turnaround time* sensitive but have an insatiable hunger for compute cycles, consuming every available cycle for years on end. These latter jobs often have turnaround times on the order of weeks or months. The concept of *cycle stealing*, popularized by systems such as Condor, handles such situations well and enables systems to run low priority, preemptible jobs whenever something more pressing is not running. These other systems are often employed on compute farms of desktops where the jobs must vacate anytime interactive system use is detected.

Maui's QoS-based preemption system allows a dedicated, non-interactive cluster to be used in much the same way. Certain QoS's may be marked with the flag **PREEMPTOR**, others with the flag **PREEMPTEE**. With this configuration, low priority, preemptee jobs can be started whenever idle resources are available. These jobs will be allowed to run until a preemptor job arrives, at which point the preemptee job will
be checkpointed if possible and vacated. This allows near immediate resource access for the preemptor job.
Using this approach, a cluster can maintain near 100% system utilization while still delivering excellent
turnaround time to the jobs of greatest value.

Use of the preemption system need not be limited to controlling low priority jobs. Other uses include optimistic scheduling and development job support.

Example:

```
----
QOSCFG[high] FLAGS=PREEMPTOR
QOSCFG[med]
QOSCFG[low]  FLAGS=PREEMPTEE
----
```

See Also: N/A .

9.0 Evaluating System Performance - Statistics, Profiling, Testing, and Simulation

---

9.1 Maui Performance Evaluation Overview

Under Construction

9.2 Job and System Statistics

Maui maintains a large number of statistics and provides several commands to allow easy access to and helpful consolidation of this information. These statistics are of three primary types:

9.2.1 Real Time Statistics

Maui provides real time statistical information about how the machine is running from a scheduling point of view. The showstats commands is actually a suite of commands providing detailed information on an overall scheduling basis as well as a per user, group, account and node basis. This command gets its information from in memory statistics which are loaded at scheduler start time from the scheduler checkpoint file. (See the Checkpoint Overview for more information) This checkpoint file is updated from time to time and when the scheduler is shutdown allowing statistics to be collected over an extended timeframe. At any time, real time statistics can be reset using the resetstats command.

In addition to the showstats command, the showgrid command also obtains its information from the in memory stats and checkpoint file. This command display 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. See the showgrid command documentation for more information.

9.2.2 Profiling Historical Usage

Historical usage information can be obtained for a specific timeframe, class of jobs, and/or portion of resources using the profiler command. This command operates on the detailed job trace information recorded at the completion of each job. These traces are stored in the directory pointed to by the STATDIR parameter which defaults to **$(MAUIHOMEDIR)/stats**. Within this directory, statistics files are maintained using the format `WWW_MMM_DD_YYYY` (i.e, Mon_Jul_16_2001) with jobs traces being recorded in the file associated with the day the job completed. Each job trace is white space delimited flat text and may be viewed directly with any text reader.

When profiling statistics, stat files covering the time frame of interest should be aggregated into a single file. This file can be passed to the profiler command along with a number of configuration flags controlling what data should be processed and how it should be display.

Command line flags allow specification of constraints such as earliest start date, or latest completion date. Flags can also be used to evaluate only jobs associated with specific users, groups, accounts, or QOS's. Further, it is possible to specify that only jobs run on certain nodes be processed. Because the trace files are flat text, simple UNIX text processing tools such as awk, sed, or grep can be used to create more elaborate filters should they be needed.

The output of the profiler command provides extensive detailed information about what jobs ran and what level of scheduling service they received. The profiler command documentation should be consulted for more information.

9.2.3 FairShare Usage Statistics

Regardless of whether of not 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> (i.e., FS.982713600) with one file created per fairshare window. (See the Fairshare Overview for more information) These files are also flat text and record credential based usage statistics. Information from these files can be seen via the diagnose -f command.

See Also:

Simulation Overview

SMP Aspects

Fairness Policies
Prioritization
Resource Allocation
Policies

Shared vs Dedicated

SMP nodes are often used to run jobs which do not use all available resources on that node. How Maui handles these unused resources is controlled by the parameter NODEACCESSPOLICY. If this parameter is set to SHARED, Maui will allow tasks of other jobs to use the resources. If this parameter is set to DEDICATED, Maui will mark these resources unavailable for use by other jobs.

Reservations

Diagnosing System Behavior/Problems

Maui provides a number of commands for diagnosing system behavior. Scheduling in a complicated task and oftentimes a scheduler will behave exactly as you said, which may not be exactly what you want! Diagnosing thus includes both looking for system failures as well

as determining current functioning system behavior. Quite often, problems may be corrected through configuration changes which more accurately reflect a site's desires.

When diagnosing system problems, the diagnose command may become your best friend. 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 diagnose command provide in depth analysis of the following objects and subsystems

Object/Subsystem	Diagnose Flag	Use
Account	-a	shows detailed account configuration information
FairShare	-f	shows detailed fairshare configuration information as well as current fairshare usage
Frame	-m	shows detailed frame information
Group	-g	shows detailed group information
Job	-j	shows detailed job information. Reports on corrupt job attributes, unexpected states, and excessive job failures
Node	-n	shows detailed node information. Reports on unexpected node states and resource allocation conditions.
Partition	-t	shows detailed partition information
Priority	-p	shows detailed job priority information including priority factor contributions to all idle jobs
QOS	-Q	shows detailed QOS information
Queue	-q	indicates why ineligible jobs or not allowed to run
Reservation	-r	shows detailed reservation information. Reports on reservation corruption of unexpected reservation conditions
User	-u	shows detailed user information

Additionally, the checkjob and checknode routines provide detailed information and sanity checking on individual jobs and nodes respectively.

Using Maui Logs for Troubleshooting

Maui logging is extremely useful in determining the cause of a problem. Where other systems may be cursed for not providing adequate logging to diagnose a problem, Maui may be cursed for the opposite reason. If the logging level is configured too high, huge volumes of log output may 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 the Logging Overview for further information.

Using a Debugger

If other methods do not resolve the problem, the use of a debugger can provide missing information. While output recorded in the Maui logs can specify which routine is failing, the debugger can actually locate the very source of the problem. Log information can help you pinpoint exactly which section of code needs to be examined and which data is suspicious. Historically, combining log information with debugger flexibility have made locating and correcting Maui bugs a relatively quick and straightforward process.

To use a debugger, you can either *attach* to a running Maui process or start Maui under the debugger. Starting Maui under a debugger requires that the MAUIDEBUG environment variable be set to the value 'yes' to prevent Maui from daemonizing and backgrounding itself. The following example shows a typical debugging start up using gdb.

```
----
> export MAUIDEBUG=yes
> cd <MAUIHOMEDIR>/src
> gdb ../bin/maui
> b QOSInitialize
> r
----
```

The gdb debugger has the ability to specify conditional breakpoints which make debugging much easier. For debuggers which do not have such capabilities, the 'TRAP*' parameters are of value allowing breakpoints to be set which only trigger when specific routines are processing particular nodes, jobs or reservations. See the TRAPNODE, TRAPJOB, TRAPRES, and TRAPFUNCTION parameters for more information.

Controlling behavior after a 'crash'
Setting 'CRASHMODE'

See also:

Troubleshooting Individual Jobs.

9.3 Profiling Current and Historical Usage

Under Construction

9.4 Testing New Versions and Configurations

Under Construction

9.5 Answering 'What If?' Questions with the Simulator

Under Construction, see 16.0 Simulations.

10.0 Managing Shared Resources - SMP Issues and Policies

- 10.1 Consumable Resource Handling
- 10.2 Load Balancing Features

10.1 Consumable Resource Handling

Maui is designed to inherently handle consumable resources. Nodes possess resources, and workload (jobs) consume resources. Maui tracks any number of consumable resources on a per node and per jobs basis. Work is under way to allow 'floating' per system resources to be handled as well. When a job is started on a set of nodes, Maui tracks how much of each available resource must be dedicated to the tasks of the job. This allows Maui to prevent per node oversubscription of any resource, be it CPU, memory, swap, local disk, etc.

Recent enhancements to Loadleveler (version 2.2 and above) finally provide a resource manager capable of exercising this long latent capability. These changes allow a user to specify per task consumable resources and per node available resources. For example, a job may be submitted requiring 20 tasks, with 2 CPUs and 256 MB per task. Thus, Maui would allow a node with 1 GB of Memory and 16 processors to allow run 4 of these tasks because 4 tasks would consume all of the available memory. Consumable resources allow more intelligent allocation of resources allowing better management of shared node resources.

No steps are required to enable this capability, simply configure the underlying resource manager to support it and Maui will pick up this configuration.

10.2 Load Balancing Features

Under Construction

11.0 General Job Administration

- 11.1 Job Holds
- 11.2 Job Priority Management
- 11.3 Suspend/Resume Handling
- 11.4 Checkpoint/Restart Facilities

11.1 Job Holds

Holds and Deferred Jobs

A job *hold* is a mechanism by which a job is placed in a state where it is not eligible to be run. Maui supports job holds applied by users, admins, and even resource managers. These holds can be seen in the output of the showq and checkjob commands. A job with a hold placed on it cannot be run until the hold is removed. If a hold is placed on a job via the resource manager, this hold must be released by the resource manager provided command (i.e., llhold for Loadleveler, or qhold for PBS).

Maui supports two other types of holds. The first is a temporary hold known as a 'defer'. A job is deferred if the scheduler determines that it cannot run. This can be because it asks for resources which do not currently exist, does not have allocations to run, is rejected by the resource manager, repeatedly fails after start up, etc. Each time a job gets deferred, it will stay that way, unable to run for a period of time specified by the DEFERTIME parameter. If a job appears with a state of deferred, it indicates one of the previously mentioned failures has occurred. Details regarding the failure are available by issuing the 'checkjob <JOBID>' command. Once the time specified by DEFERTIME has elapsed, the job is automatically released and the scheduler again attempts to schedule it. The 'defer' mechanism can be disabled by setting DEFERTIME to '0'. To release a job from the defer state, issue 'releasehold -a <JOBID>'.

The second 'Maui-specific' type of hold is known as a 'batch' hold. A batch hold is only applied by the scheduler and is only applied after a serious or repeated job failure. If a job has been deferred and released DEFERCOUNT times, Maui will place it in a batch hold. It will remain in this hold until a scheduler admin examines it and takes appropriate action. Like the defer state, the causes of a batch hold can be determined via checkjob and the hold can be released via releasehold.

Like most schedulers, Maui supports the concept of a job hold. Actually, Maui supports four distinct types of holds, *user* holds, *system* holds, *batch* holds, and *defer* holds. Each of these holds effectively block a job, preventing it from running, until the hold is removed.

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 which basically tells the scheduler not to run the job while the hold is in place. The user may utilize this capability because the job's data is not yet ready, or he wants to be present when the job runs so as to monitor results. Such user holds are created by, and under the control of a non-privileged and may be removed at any time by that user. As would be expected, users can only place holds on their jobs. Jobs with a user hold in place will have a Maui state of **Hold** or **UserHold** depending on the resource manager being used.

System Holds

The second category of hold is the system hold. This hold is put in place by a system administrator 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 administrator can place and release system holds on any job regardless of job ownership. However, unlike a user hold, a normal user cannot release a system hold even on his 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 Maui state of **Hold** or **SystemHold** depending on the resource manager being used.

Batch Holds

Batch holds constitute the third category of job holds. These 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. Some of the possible reasons are listed below:

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 which 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 which are placed in a batch hold will show up within Maui in the state **BatchHold**.

Job Defer

In most cases, a job violating these policies will not be placed into a batch hold immediately. Rather, it will be deferred. The parameter DEFERTIME indicates how long it will be deferred. At this time, it will be 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 will be released and deferred up to DEFERCOUNT times at which point the scheduler places a batch hold on the job and waits for a system administrator to determine the correct course of action. Deferred jobs will have a Maui 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 Maui logs should provide detailed information about the cause of any batch hold or job deferral.

NOTE: As of Maui 3.0.7, the reason a job is deferred or placed in a batch hold is stored in memory but is not checkpointed. Thus this info is available only until Maui is recycled at which point the checkjob command will no longer display this 'reason' info.

(under construction)

Controlling Backfill Reservation Behavior
- Reservation Thresholds
- Reservation Depth

Resource Allocation Method
- First Available
- Min Resource
- Last Available

WallClock Limit
- Allowing jobs to exceed wallclock limit
  - MAXJOBOVERRUN
- Using Machine Speed for WallClock limit scaling
  - USEMACHINESPEED

Controlling Node Access
- NODEACCESSPOLICY

11.2 Job Priority Management

Under Construction

11.3 Suspend/Resume Handling

Under Construction

11.4 Checkpoint/Restart Facilities

Under Construction

12.0 General Node Administration

Since Maui interoperates with a number of resource managers of varying capabilities, it must possess a somewhat redundant set of mechanisms for specifying node attribute, location, and policy information. Maui determines a node's configuration through one or more of the following approaches:

- Direct resource manager specification

Some node attribute may be directly specified through the resource manager. For example, Loadleveler allows a site to assign a 'MachineSpeed' value to each node. If the site chooses to specify this value within the Loadleveler configuration, Maui will obtain this info via the Loadleveler scheduling API and use it in scheduling decisions. The list of node attributes supported in this manner varies from resource manager to resource manager and should be determined by consulting resource manager documentation.

- Translation of resource manager specified 'opaque' attributes

Many resource managers support the concept of opaque node attributes, allowing a site 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 Maui parameters, sites can assign a meaning within Maui to these opaque node attributes and extract specific node information. For example, setting the parameter 'FEATUREPROCSPEEDHEADER xps' will cause a node with the opaque string 'xps950' to be a assigned a processor speed of 950 MHz within Maui.

- Default node attributes

Some default node attributes can be assigned on a frame or partition basis. Unless explicitly specified otherwise, nodes within the particular node or partition will be assigned these default attribute values. See the Partition Overview for more information.

- Direct maui parameter specification

Maui also provides a parameter named NODECFG which allows direct 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 may wish to specify something like the following:

```
NODECFG[node031] MAXJOB=2 PROCSPEED=600 PARTITION=small
```

These approaches may be mixed and matched according to the site's local needs.

Precedence for the approaches generally follows the order listed above in cases where conflicting node configuration information is specified through one or more mechanisms.

- 12.1 Node Location (Partitions, Frames, Queues, etc.)
- 12.2 Node Attributes (Node Features, Speed, etc.)
- 12.3 Node Specific Policies (MaxJobPerNode, etc.)

12.1 Node Location

Nodes can be assigned three types of location information based on partitions, frames, and/or queues.

12.1.1 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 a 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 the Partition Overview for more information.

12.1.2 Frames

Frame based location information is orthogonal to the partition based configuration and is mainly an organizational construct. In general frame based location usage, a node is assigned both a frame and a slot number. This approach has descended from the IBM SP2 organizational approach in which a frame can contain any number of slots but typically contains between 1 and 64. Using the frame and slot number combo, individual compute nodes can be grouped and displayed in a more ordered manner in certain Maui commands (i.e., showstate). Currently, frame 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 manually specified via the NODECFG parameter.

Example:

```
---
# maui.cfg

NODECFG[node024] FRAME=1 SLOT=1
NODECFG[node025] FRAME=1 SLOT=2
NODECFG[node026] FRAME=2 SLOT=1 PARTITION=special
...
```

When specifying node and frame information, slot values must be in the range of 1 to 32 (limited to 1 to 16 in Maui 3.0 and earlier). and frames must be in the range of 1 to 64.

12.1.3 Queues

Some resource managers allow queues (or classes) to be defined and then associated with a subset of available compute resources. With such systems, such as Loadleveler or PBSPro, these queue to node mappings are automatically detected. On resource managers which do not provide this service, Maui provides alternative mechanisms for enabling this feature.

12.1.3.1 OpenPBS Queue to Node Mapping

Under OpenPBS, queue to node mapping can be accomplished setting the queue `acl_hosts` parameter to the mapping hostlist desired within PBS. Further, the `acl_host_enable` parameter should be set to `False`. **NOTE**: Setting `acl_hosts` and then setting `acl_host_enable` to `True` will constrain the list of hosts from which jobs may be submitted to the queue. Prior to Maui 3.0.7p3, queue to node mapping was only enabled when `acl_host_enable` was set to `True`, thus, for these versions, the `acl_host` list should always include all submission hosts.

12.2 Node Attributes

Nodes can possess a large number of attributes describing their configuration. 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 below.

NODETYPE

The **NODETYPE** attribute is most commonly used in conjunction with an allocation management system such as QBank. In these cases, each node is assigned a node type and within the allocation management system, each node type is assigned a charge rate. For example, a site may wish to charge users more for using large memory nodes and may assign a node type of 'BIGMEM' to these nodes. The allocation management system would then charge a premium rate for jobs using BIGMEM nodes. (See the Allocation Manager Overview for more information.)

Node types are specified as simple strings. If no node type is 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.

Example:

```
---
# maui.cfg

NODECFG[node024] NODETYPE=BIGMEM
...
---
```

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. Maui's Node Set scheduling policies allow a site 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.

SPEED

A node's speed is very similar to its procspeed but is specified as a relative value. In general use, the speed of a base node is determined and assigned a speed of 1.0. A node that is 50% faster would be assigned a value of 1.5 while a slower node may receive a value which is proportionally less than 1.0. Node speeds do not have to be directly proportional to processor speeds and may take into account factors such as memory size or networking interface. Generally, node speed information is used to determine proper wallclock limit and CPU time scaling adjustments.

Node speed information is specified as a unitless floating point ratio and can be specified through the resource manager or with the NODECFG parameter.

The **SPEED** specification must be in the range of 0.01 to 100.0.

FEATURES

Not all resource managers allow specification of opaque node features. For these systems, the NODECFG parameter can be used to directly assign a list of node features to individual nodes.

12.3 Node Specific Policies

Specification of node policies is fairly limited within Maui mainly because the demand for such policies is limited. These policies allow a site to specify on a node by node basis what the node will and will not support. Node policies may be applied to specific nodes or applied system wide using the specification '`NODECFG[DEFAULT] ...`' Note that these policies were introduced over time so not all policies are supported in all versions.

MAXJOB (Maui 3.0.7 and higher)

This policy constrains the number of total independent jobs a given node may run simultaneously. It can only be specified via the **NODECFG** parameter.

MAXJOBPERUSER (Maui 3.0.7 and higher)

This policy constrains the number of total independent jobs a given node may run simultaneously associated with any single user. Like **MAXJOB**, it can only be specified via the NODECFG parameter.

MAXLOAD (Maui 3.2.2 and higher)

MAXLOAD constrains the CPU load the node will support as opposed to the number of jobs. If the node's load exceeds the **MAXLOAD** limit and the NODELOADPOLICY parameter is set to **ADJUSTSTATE** , the node will be marked busy. Under Maui 3.0, the max load policy could be applied system wide using the parameter NODEMAXLOAD.

Node policies are used strictly as constraints. If a node is defined as having a single processor or the NODEACCESSPOLICY is set to **DEDICATED**, and a **MAXJOB** policy of 3 is specified, Maui will probably not run more than one job per node. A node's configured processors must be specified so that multiple jobs may 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, 'np' per node in the PBS 'nodes' file.

Example:

```
---
# maui.cfg

NODECFG[node024] MAXJOB=4 MAXJOBPERUSER=2
NODECFG[node025] MAXJOB=2
NODECFG[node026] MAXJOBPERUSER=1
NODECFG[DEFAULT] MAXLOAD=2.5
...
---
```

Also See:

<N/A>

13.0 Resource Managers and Interfaces

13.1 Resource Manager Overview

Maui requires the services of a resource manager in order to properly function. This resource manager provides information about the state of compute resources (nodes) and workload (jobs). Maui also depends on the resource manager to manage jobs, instructing it when to start and/or cancel jobs.

Maui can be configured to manage one or more resource managers simultaneously, even resource managers of different types. However, migration of jobs from one resource manager to another is not currently allowed meaning jobs submitted onto one resource manager cannot run on the resources of another.

13.1.1 Scheduler/Resource Manager Interactions

Maui interacts with all resource managers in the same basic format. Interfaces are created to translate Maui concepts regarding workload and resources into native resource manager objects, attributes, and commands.

Information on creation a new scheduler resource manager interface can be found in the Adding New Resource Manager Interfaces section.

13.1.1.1 Resource Manager Commands

In the simplest configuration, Maui interacts with the resource manager using the four primary functions listed below:

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, Maui 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 overviews. (LL, PBS, or WIKI).

In addition to these base commands, other commands are required to support advanced features such a dynamic job support, suspend/resume, gang scheduling, and scheduler initiated checkpoint/restart.

13.1.1.2 Resource Manager Flow

Early versions of Maui (i.e., Maui 3.0.x) interacted with resource managers in a very basic manner stepping through a serial 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 information (optional)
5. cancel jobs which violate policies
6. start jobs in accordance with available resources and policy constraints
7. handle user commands
8. repeat

Each step would complete before the next step started. As systems continued to grow in size and complexity however, it became apparent that the serial model described above would not work. Three primary motivations drove the effort to replace the serial model with a concurrent threaded approach. These motivations were reliability, concurrency, and responsiveness.

Reliability

A number of the resource managers Maui interfaces to were unreliable to some extent. This resulted in calls to resource management API's with exitted or crashed taking the entire scheduler with them. Use of a threaded approach would cause only the calling thread to fail allowing the master scheduling thread to recover. Additionally, a number of resource manager calls would hang indefinately, locking up the scheduler. These hangs could likewise be detected by the master scheduling thread and handled appropriately in a threaded environment.

Concurrency

As resource managers grew in size, the duration of each API global query call grew proportionally. Particularly, queries which required contact with each node individually became excessive as systems grew into the thousands of nodes. A threaded interface allowed the scheduler to concurrently issue multiple node queries resulting in much quicker aggregate RM query times.

Responsiveness

Finally, in the non-threaded serial approach, the user interface was blocked while the scheduler updated various aspects of its workload, resource, and queue state. In a threaded model, the scheduler could continue to respond to queries and other commands even while fresh resource manager state information was being loaded resulting in much shorter average response times for user commands.

Under the threaded interface, all resource manager information is loaded and processed while the user interface is still active. Average aggregate resource manager API query times are tracked and new RM updates are launched so that the RM query will complete before the next scheduling iteration should start. Where needed, the loading process uses a *pool* of worker threads to issue large numbers of node specific information queries concurrently to accelerate this process. The master thread continues to respond to user commands until all needed resource manager information is loaded **and** either a scheduling relevant *event* has occurred **or** the scheduling iteration time has arrived. At this point, the updated information is integrated into Maui's state information and scheduling is performed.

13.1.2 Resource Manager Specific Details (Limitations/Special Features)

(Under Construction)

LL/LL2
PBS
Wiki

Synchronizing Conflicting Information
Maui does not trust resource manager. All node and job information is reloaded on each iteration. Discrepancies are logged and handled where possible.
NodeSyncDeadline/JobSyncDeadline overview.

Purging Stale Information

Thread

See Also:

13.2 Resource Manager Configuration

The type of resource manager to interface to is specified using the RMTYPE parameter. This parameter takes an argument of the form **<RMTYPE>[:<RMSUBTYPE>]**. Currently, the following resource manager types are supported:

LL2 - Loadleveler version 2.1 and 2.2
PBS - OpenPBS and PBSPro (all versions)
WIKI - Text based API used by LRM, YRM, BProc, and other resource managers
SGE - Sun's Grid Engine Resource Manger

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. As noted above, Maui can support more than one resource manager simultaneously. Consequently, all resource manager parameters are specified as arrays. For example, to interface to the Loadleveler scheduling API, one would specify

```
RMTYPE[0] LL2
```

See the Parameters Overview for more information about parameter specification.

In addition to **RMTYPE**, other parameters allow further control of the scheduler/resource manager interface. These include **RMNAME**, **RMPORT**, **RMHOST**, **RMTIMEOUT**, **RMAUTHTYPE**, **RMCONFIGFILE**, and **RMNMPORT**.

The **RMNAME** parameter allows a site to associate a name with a particular resource manager so as to simplify tracking of this interface within Maui. To date, most sites have chosen to setup only one resource manager per scheduler making this parameter largely unnecessary. **RMHOST** and **RMPORT** allow specification of where the resource manager is located. These parameters need only to be specified for resource managers using the WIKI interface or with PBS when communication with a non-default server is required. In all other cases, the resource manager is automatically located.

The maximum amount of time Maui will wait on a resource manager call can be controlled by the **RMTIMEOUT** parameter which defaults to 30 seconds. Only rarely will this parameter need to be changed. **RMAUTHTYPE** allows specification of how security over the scheduler/resource manager interface is to be handled. Currently, only the WIKI interface is affected by this parameter. The allowed values are documented in the RMAUTHTYPE parameter description.

Another RM-specific parameter is **RMCONFIGFILE**, which specifies the location of the resource manager's primary config file and is used when detailed resource manager information not available via the scheduling interface is required. It is currently only used with the Loadleveler interface and needs to only be specified when using Maui meta-scheduling capabilities.

Finally, the RMNMPORT 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. It is currently only used within PBS to allow MOM specific information to be gathered and utilized by Maui.

13.1.2 Scheduler/Resource Manager Interactions

In the simplest configuration, Maui interacts with the resource manager using the four primary functions listed below:

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, Maui 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 overviews. (LL, PBS, or WIKI).

In addition to these base commands, other commands are required to support advanced features such a dynamic job support, suspend/resume, gang scheduling, and scheduler initiated checkpoint restart. More information about these commands will be forthcoming.

Information on creation a new scheduler resource manager interface can be found in the Adding New Resource Manager Interfaces section.

13.3 Resource Manager Extensions

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 which 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, Maui 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.

How this is done varies with the resource manager. Both Loadleveler and Wiki allow the specification of a *comment* field. (In Loadleveler, specified as '`#@comment="<X>"`') PBS does not support this ability by default but is extendible via the '**-W**' flag. (see the PBS Resource Manager Extension Overview)

Using the resource manager specific method, the following job extensions are currently available:

Name	Format	Default Value	Description	Example
ACCESSMODE	one of **DEDICATED** or **SHARED**	**SHARED**		`ACCESSMODE:DEDICATED`
DMEM	<INTEGER>	0	dedicated memory per task in MB	`DMEM:512`
FLAGS	one or more of the following comma separated keywords **ADVRES[:RESID]**, **RESTARTABLE**, **PREEMPTEE**, **PREEMPTOR**, **NOQUEUE**	[NONE]	associates various flags with the job	`FLAGS:ADVRES`

HOSTLIST	comma delimited list of hostnames	[NONE]	indicates a exact set, superset, or subset of nodes on which the job must run	`HOSTLIST:nodeA,nodeB,nodeE`
NODESET	<SETTYPE>:<SETATTR>[:<SETLIST>]	[NONE]	specifies nodeset constraints for job resource allocation. (See the NodeSet Overview for more information)	`NODESET:ONEOF:PROCSPEED:350,400,450`
PARTITION	<STRING>[:<STRING>]...	[NONE]	specifies the partition (or partitions) in which the job must run. **NOTE**: the job must have access to this partition based on system wide or credential based partition	`PARTITION:math:geology` (The job must only run in the `math` partition or the `geology` partition)

			access lists.	
QOS	<STRING>	[NONE]		`QOS:highprio`
QUEUEJOB	one of `FALSE` or `TRUE`	`TRUE`	Indicates whether or not the scheduler should queue the job if resources are not available to run the job immediately	`QUEUEJOB:FALSE`
SGE	<WINDOWCOUNT>:<DISPLAYNAME>	[NONE]		`SGE:8:pinky`
SID	<STRING>	[NONE]		`SID:silverA`
TPN	<INTEGER>	0		`TPN:4`
TRL	<INTEGER>[,<INTEGER>]...	0		`TRL:2,4,8,16`

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>]...`'

See the following examples:

Example 1

```
# Loadleveler command file
#@comment="HOSTLIST:node1,node2;QOS:special;SID:silverA"
```

Job must run on nodes `node1` and `node2` using the QoS `special`. The job is also associated with the system id `silverA` allowing the silver daemon to monitor and control the job.

Example 2

```
# PBS command file
```

```
# PBS -W X="NODESET=ONEOF:NETWORK;DMEM:64"
```

Job will have resources allocated subject to network based nodeset constraints. Further, each task will dedicate 64 MB of memory.

Example 3

```
# qsub -l nodes=4,walltime=1:00:00 -W x="FLAGS:ADVRES:john.1"
```

Job will be forced to run within the `john.1` reservation.

See Also:

Resource Manager Overview

13.4 Adding New Resource Manager Interfaces

Maui currently interfaces with about 6 different resource manager systems. Some of these interact through a resource manager specific interface (ie, OpenPBS/PBSProc, Loadleveler) while others interact through a simple text based interfaces known as Wiki. (see the Wiki Overview). 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 Maui interface would require more changes to Maui mods. If a scheduling API already exists within the resource manager, creation a a resource manager specific Maui interface is often selected.

Regardless of the interface approach selected, adding support for a new resource manager is typically a straight forward 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 distinctive and unique concepts which must be addressed.

13.4.1 Resource Manager Specific Interfaces

If the resource manger specific interface is desired, then typically a scheduling API library/header file combo is required. (i.e., for PBS, libpbs.a + pbs_ifl.h, etc.) This resource manager provided API provides calls which can be linked into Maui to obtain the 'raw' resource manager data including both jobs and compute nodes. Additionally, this API should provide policy information about the resource manager configuration if it is desired that such policies be specified via the resource manager rather than the scheduler and that Maui know of and respect these policies. The new '<X>Interface.c' module would be responsible for loading information from the resource manager, translating this information, and then populating the appropriate Maui data structures. The existing **LLInterface.c**, **PBSInterface.c** and **WikiInterface.c** modules provide templates indicating how to do this.

The first step in this process is defining the new resource manager type. This is accomplished by modifying **maui_struct.h** and **maui_global.h** header files to define the new RMTYPE parameter value. With this defined, the RMInterface.c module must be modified to call the appropriate resource manager specific calls which will eventually be created within the '<X>Interface,c' module. This process is quite easy and involves merely extending existing resource manager specific case statements within the general resource manager calls.

The vast majority of the development effort in entailed in creating the resource manager

specific data collection and job management calls. These calls populate Maui data structures, and are responsible for passing Maui scheduling commands on to the resource manager. The base commands are GetJobs, GetNodes, StartJob, and CancelJob but if the resource manager support is available, extended functionality can be enabled by creating commands to suspend/resume jobs, checkpoint/restart jobs, and/or allow support of dynamic jobs.

If the resource manager provides a form of event driven scheduling interface, this will also need to be enabled. The **PBSInterface.c** module provides a template for enabling such an interface within the PBSProcessEvent() call.

13.4.2 Wiki Interface

The Wiki interface is a good alternative if the resource manager does not already support some form of existing scheduling API. For the most part, use of this API requires the same amount of effort as creating a resource manager specific interface but development effort focussed within the resource manager. Since Wiki is already defined as a resource manager type, no modifications are required within Maui. 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 the Wiki Overview. Additionally, resource manager parameters must be created to allow a site to configure this interface appropriately.

Efforts are currently underway to create a new XML based interface with an improved transport and security model. This interface will also add support for more flexible resource and workload descriptions as well as resource manager specific policy configuration. It is expected that this interface will be available in mid 2002.

14.0 Trouble Shooting and System Maintenance

14.1 Internal Diagnostics

Under Construction

14.2 Logging Overview

The Maui Scheduler provides the ability to produce detailed logging of all of its activities. The LOGFILE and/or LOGDIR parameters within the maui.cfg file specify the destination of this logging information. Logging information will be written in the file <MAUIHOMEDIR>/<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 '**maui.log**' respectively, resulting in scheduler logs being written to **<MAUIHOMEDIR>/log/maui.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 will control the number of old logs maintained and defaults to 1. Rolled log files will have a numeric suffix appended indicating their order.

The parameter LOGLEVEL controls the verbosity of the information. Currently, LOGLEVEL values between 0 and 9 are used to control the amount of information logged, with 0 being the most terse, logging only the most server problems detected, while 9 is the most verbose, commenting on just about everything. The amount of information provided at each log level is approximately an order of magnitude greater than what is provided at the log level immediately below it. A LOGLEVEL of 2 will record virtually all critical messages, while a log level of 4 will provide general information describing all actions taken by the scheduler. If a problem is detected, you may wish 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 will result 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. NOTE: If high log levels are desired for an extended period of time and your Maui home directory is located on a network filesystem, 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 the parameters documentation.

The logging that occurs is of five major types, subroutine information, status information, scheduler warnings, scheduler alerts, and scheduler errors. These are described in detail below:

1.Subroutine Information. Each subroutine is logged, along with all printable parameters. Major subroutines are logged at lower LOGLEVELs while all subroutines are logged at higher LOGLEVELs. Example:

CheckPolicies(fr4n01.923.0,2,Reason)

2.Status Information. Information about internal status is logged at all LOGLEVELs. Critical internal status is indicated at low LOGLEVELs while less critical and voluminous status information is logged at higher LOGLEVELs. Example:

INFO: Job fr4n01.923.0 Rejected (Max User Jobs)
INFO: Job[25] 'fr4n01.923.0' Rejected (MaxJobPerUser Policy Failure)

3.Scheduler Warnings. Warnings are logged when the scheduler detects an unexpected value or receives an unexpected result from a system call or subroutine. They are not necessarily indicative of problems and are not catastrophic to the scheduler. Example:

WARNING: cannot open fairshare data file '/home/loadl/maui/stats/FS.87000'

4.Scheduler Alerts. Alerts are logged when the scheduler detects events of an unexpected nature which may indicate problems in other systems or in objects. They are typically of a more severe nature than are warnings and possibly should be brought to the attention of scheduler administrators. Example:

ALERT: job 'fr5n02.202.0' cannot run. deferring job for 360 Seconds

5.Schedulers Errors. Errors are logged when the scheduler detects problems of a nature of which it is not prepared to deal. It will try to back out and recover as best it can, but will not necessarily succeed. Errors should definitely be be monitored by administrators. Example:

ERROR: cannot connect to Loadleveler API

On a regular basis, use the command grep -E "WARNING|ALERT|ERROR" maui.log to get a listing of the problems the scheduler is detecting. On a production system working normally, this list should usually turn up empty. 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 Maui Scheduler (before the configuration file is read) maui 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 will be recorded.

If problems are detected in the use of one of the client commands, the client command can be re-issued with the -L <LOGLEVEL> command line arg specified. This argument causes debug information to be logged to STDERR as the client command is running. Again, <LOGLEVEL> values from 0 to 9 are
supported.

In addition to the log file, the Maui Scheduler reports all events it determines to be critical to the UNIX syslog facility via the 'daemon' facility using priorities ranging from 'INFO' to

'ERROR'. This logging is not affected by LOGLEVEL. In addition to errors and critical events, all user commands that affect the state of the jobs, nodes, or the scheduler are also logged via syslog.

The logging information is extremely helpful in diagnosing problems, but it can also be useful if you are simply trying to become familiar with the "flow" of the scheduler. The scheduler can be run with a low LOGLEVEL value at first to show the highest level functions. This shows high-level data and control flow. Increasing the LOGLEVEL increases the number of functions displayed as familiarity with the scheduler flow grows.

The LOGLEVEL can be changed "on-the-fly" by use of the changeparam command, or by modifying the maui.cfg file and sending the scheduler process a SIGHUP. Also, if the scheduler appears to be "hung" or is not properly responding, the LOGLEVEL can be incremented by one by sending a SIGUSR1 signal to the scheduler process. Repeated SIGUSR1 signals will continue to increase the LOGLEVEL. The SIGUSR2 signal can be used to decrement the LOGLEVEL by one.

If an unexpected problem does occur, save the log file as it is often very helpful in isolating and correcting the problem.

14.3 Using the Message Buffer

Under Construction

14.4 Handling Events with the Notification Routine

Maui 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 allows a site 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 which has occurred. The location of the notification program may be specified as a relative or absolute path. If a relative path is specified, Maui will look for the notification relative to the **$(MAUIHOMEDIR)/tools** directory. In all cases, Maui will verify the existence of the notification program at start up and will disable it if it cannot be found or is not executable.

The notification program's action may 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 commandline arguments in a simple <EVENTTYPE>: <MESSAGE> format. The following event types are currently enabled:

Event Type	Format	Description
BANKFAILURE	<MESSAGE>	Maui cannot successfully communicate with the bank due to reasons such as connection failures, bank corruption, or parsing failures
JOBCORRUPTION	<MESSAGE>	An active job is in an unexpected state or has one or more allocated nodes which 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	<RESNAME> <RESTYPE> <NAME> <PRESENTTIME> MSTARTTIME> <ENDTIME> <NODECOUNT>	A new reservation has been created
RESERVATIONDESTROYED	<RESNAME> <RESTYPE> <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 Maui 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>);**

See Also:

N/A

14.5 Issues with Client Commands

Client Overview:

Maui clients are implemented as symbolic links to the executable **maui_client**. When a maui client command is run, the client executable determines the name under which it is run and behaves accordingly. At the time Maui was configured, a *home directory* was specified. The Maui client will attempt to open the config file **maui.cfg** in this home directory on the node where the client command is executed. This means that the home directory specified at configure time must be available on all hosts where the maui client commands will be executed. This also means that a **maui.cfg** file must be available in this directory. When the clients open this file, they will try to load the **MAUISERVER** and **MAUIPORT** parameters to determine how to contact the Maui server.

NOTE: The home directory value specified at configure time can be overridden by creating an **/etc/maui.cfg** file or by setting the '**MAUIHOMEDIR**' environment variable.

Once the client has determined where the Maui server is located, it creates a message, adds an encrypted checksum, and sends the message to the server. Note that the Maui client and Maui server must use the same secret checksum seed for this to work. When the Maui server receives the client request and verifies the checksum, it processes the command and returns a reply.

Diagnosing Client Problems:

The easiest way to determine where client failures are occuring is to utilize built in maui logging. On the client side, use the '**-L**' flag. For example,

```
> showq -L9
```

NOTE: Maui 3.0.6 and earlier specified the desired client side logging level using the '**-D**' flag (i.e., `showq -D 9`)

This will dump out a plethora of information about loading the configfile, connecting to the maui server, sending a request, and receiving a response. Wading through this information almost always will reveal the source of the problem. If it does not, the next step is to look at the maui server side logs. The easiest way to do this is to use the following steps:

```
> schedctl -s
```
(stop Maui scheduling so that the only activity is handling maui client requests)
```
> changeparam LOGLEVEL 7
```
(set the logging level to 'very verbose')
```
> tail -f log/maui.log | more
```
(tail the maui.log activity)

(In another window)

```
> showq
```

The **maui.log** file should record the client request and any reasons it was rejected.

If these steps do not reveal the source of the problem, the next step may be to check the mailing list archives, post a question to the mauiusers list, or contact Maui support.

14.6 Tracking System Failures

Under Construction

14.7 Problems with Individual Jobs

To determine why a particular job will not start, there are several commands which can be helpful.

checkjob -v

Checkjob will evaluate the ability of a job to start immediately. Tests include resource access, node state, job constraints (ie, startdate, taskspernode, QOS, etc). Additionally, command line flags may be specified to provide further information.

```
-l <POLICYLEVEL>    // evaluate impact of throttling policies on job feasibility
-n <NODENAME>       // evaluate resource access on specific node
-r <RESERVATION_LIST>  // evaluate access to specified reservations
```

checknode

Display detailed status of node

diagnose -j

Display high level summary of job attributes and perform sanity check on job attributes/state.

diagnose -q

Display various reasons job is considered 'blocked' or 'non-queued'.

showbf -v

Determine general resource availability subject to specified constraints.

See also:

Diagnosing System Behavior/Problems

15.0 Improving User Effectiveness

15.1 User Feedback Loops

In evaluating a given system, it is interesting to note that 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 which modifies his 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 system utilization overall. 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 which help users make improved decisions. Cluster wide resource availability information and per job resource utilization information.

15.1.1 Improving Job Size/Duration Requests

Maui provides a number of informational commands which help users make improved job management decisions based on real-time cluster wide resource availability information. These commands include showbf, showgrid, and showq. Using these commands, a user can determine what resources are available, and what job configurations statistically receive the best scheduling performance.

15.1.2 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 have 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. 2) users do not no exactly what resources their jobs are utilizing 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, Maui provides the FEEDBACKPROGRAM facility. This tool allows 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.

15.2 User Level Statistics

Under Construction

15.3 Enhancing Wallclock Limit Estimates

Under Construction

15.4 Providing Resource Availability Information

Under Construction

15.5 Job Start Time Estimates

Under Construction

15.6 Collecting Performance Information on Individual Jobs

Under Construction

16.0 Simulations

16.1 Simulation Overview

16.1.1 Test Drive

If you want to see what the scheduler is capable of, the simulated test drive is probably your best bet. This allows you to safely play with arbitrary configurations and issue otherwise 'dangerous' commands without fear of losing your job! :) In order to run a simulation, you need a simulated machine (defined by a resource trace file) and a simulated set of jobs (defined by a workload trace file). Rather than discussing the advantages of this approach in gory detail up front, let's just get started and discuss things along the way.

Issue the following commands:

```
> vi maui.cfg
  (change 'SERVERMODE   NORMAL' to 'SERVERMODE   SIMULATION')
  (add 'SIMRESOURCETRACEFILE     traces/Resource.Trace1')
  (add 'SIMWORKLOADTRACEFILE  traces/Workload.Trace1')
  (add 'SIMSTOPITERATION              1')

# the steps above specified that the scheduler should do the following:
# 1)  Run in 'Simulation' mode rather than in 'Normal' or live mode.
# 2)  Obtain information about the simulated compute resources in the
#     file 'traces/Resource.Trace1'.
# 3)  Obtain information about the jobs to be run in simulation in the
#     file 'traces/Workload.Trace1'
# 4)  Load the job and node info, start whatever jobs can be started on
#     the nodes, and then wait for user commands.  Do not advance
#     simulated time until instructed to do so.

> maui &

# give the scheduler a few seconds to warm up and then look at the
# list of jobs currently in the queue.  (To obtain a full description
# of each of the commands used below, please see the command's man
# page.)

> showq

# This command breaks the jobs in the queue into three groups, 'Active'
# jobs which are currently running, 'Idle' jobs which can run as soon
# as the required resources become available, and 'Non Queued' jobs
# which are currently ineligible to be run because they violate some
# configured policy.
```

 # By default, the simulator initially submits 100 jobs from the
 # workload trace file, 'Workload.Trace1'. Looking at the 'showq'
 # output, it can be seen that the simulator was able to start 11 of
 # these jobs on the 195 nodes described in the resource trace file,
 # 'Resource.Trace1'.

 # Look at the running jobs more closely.

> showq -r

 # The output is sorted by job completion time. We can see that the
 # first job will complete in 5 minutes.

 # Look at the initial statistics to see how well the scheduler is
 # doing.

> showstats

 # Look at the line 'Current Active/Total Procs' to see current system
 # utilization.

 # Determine the amount of time associated with each simulated time
 # step.

> showconfig | grep RMPOLLINTERVAL

 # This value is specified in seconds. Thus each time we advance the
 # simulator forward one step, we advance the simulation clock forward
 # this many seconds. 'showconfig' can be used to see the current
 # value of all configurable parameters. Advance the simulator forward
 # one step.

> schedctl -S

 # 'schedctl' allows you to step forward any number of steps or to step
 # forward to a particular iteration number. You can determine what
 # iteration you are currently on using the 'showstats' command's '-v'
 # flag.

> showstats -v

 # The line 'statistics for iteration <X>' specifies the iteration you
 # are currently on. You should now be on iteration 2. This means
 # simulation time has now advanced forward <RMPOLLINTERVAL> seconds.
 # use 'showq -r' to verify this change.

> showq -r

 # Note that the first job will now complete in 4 minutes rather than
 # 5 minutes because we have just advanced 'now' by one minute. It is
 # important to note that when the simulated jobs were created both the

 # job's wallclock limit and its actual run time were recorded. The
 # wallclock time time is specified by the user indicating his best
 # estimate for an upper bound on how long the job will run. The run
 # time is how long the job actually ran before completing and
 # releasing its allocated resources. For example, a job with a
 # wallclock limit of 1 hour will be given the need resources for up to
 # an hour but may complete in only 20 minutes.

 # The output of 'showq -r' shows when the job will complete if it runs
 # up to its specified wallclock limit. In the simulation, jobs actually
 # complete when their recorded 'runtime' is reached. Let's look at
 # this job more closely.

> checkjob fr8n01.804.0

 # We can wee that this job has a wallclock limit of 5 minutes and
 # requires 5 nodes. We can also see exactly which nodes have been
 # allocated to this job. There is a lot of additional information
 # which the 'checkjob' man page describes in more detail.

 # Let's advance the simulation another step.

> schedctl -S

 # Look at the queue again to see if anything has happened.

> showq -r

 # No surprises. Everything is one minute closer to completion.

> schedctl -S

> showq -r

 # Job 'fr8n01.804.0' is still 2 minutes away from completing as
 # expected but notice that both jobs 'fr8n01.191.0' and
 # 'fr8n01.189.0' have completed early. Although they had almost 24
 # hours remaining of wallclock limit, they terminated. In reality,
 # they probably failed on the real world system where the trace file
 # was being created. Their completion freed up 40 processors which
 # the scheduler was able to immediately use by starting two more
 # jobs.

 # Let's look again at the system statistics.

> showstats

 # Note that a few more fields are filled in now that some jobs have
 # completed providing information on which to generate statistics.

 # Advance the scheduler 2 more steps.

> schedctl -S 2I

 # The '2I' argument indicates that the scheduler should advance '2'
 # steps and that it should (I)gnore user input until it gets there.
 # This prevents the possibility of obtaining 'showq' output from
 # iteration 5 rather than iteration 6.

> showq -r

 # It looks like the 5 processor job completed as expected while
 # another 20 processor job completed early. The scheduler was able
 # to start another 20 processor job and five serial jobs to again
 # utilize all idle resources. Don't worry, this is not a 'stacked'
 # trace, designed to make the Maui scheduler appear omniscient.
 # We have just gotten lucky so far and have the advantage of a deep
 # default queue of idle jobs. Things will get worse!

 # Let's look at the idle workload more closely.

> showq -i

 # This output is listed in priority order. We can see that we have
 # a lot of jobs from a small group of users, many larger jobs and a
 # few remaining easily backfillable jobs.

 # let's step a ways through time. To speed up the simulation, let's
 # decrease the default LOGLEVEL to avoid unnecessary logging.

> changeparam LOGLEVEL 0

 # 'changeparam' can be used to immediately change the value of any
 # parameter. The change is only made to the currently running Maui
 # and is not propagated to the config file. Changes can also be made
 # by modifying the config file and restarting the scheduler or
 # issuing 'schedctl -R' which forces the scheduler to basically
 # recycle itself.

 # Let's stop at an even number, iteration 60.

> schedctl -s 60I

 # The '-s' flag indicates that the scheduler should 'stop at' the
 # specified iteration.

> showstats -v

 # This command may hang a while as the scheduler simulates up to
 # iteration 60.

 # The output of this command shows us the 21 jobs have now completed.

 # Currently, only 191 of the 195 nodes are busy. Lets find out why
 # the 4 nodes are idle.

 # First look at the idle jobs.

> showq -i

 # The output shows us that there are a number of single processor
 # jobs which require between 10 hours and over a day of time. Lets
 # look at one of these jobs more closely.

> checkjob fr1n04.2008.0

 # If a job is not running, checkjob will try to determine why it
 # isn't. At the bottom of the command output you will see a line
 # labeled 'Rejection Reasons'. It states that of the 195 nodes
 # in the system, the job could not run on 191 of them because they
 # were in the wrong state (i.e., busy running other jobs) and 4 nodes
 # could not be used because the configured memory on the node did
 # not meet the jobs requirements. Looking at the 'checkjob' output
 # further, we see that this job requested nodes with '>= 512' MB of
 # RAM installed.

 # Let's verify that the idle nodes do not have enough memory
 # configured.

> diagnose -n | grep -e Idle -e Name

 # The grep gets the command header and the Idle nodes listed. All
 # idle nodes have only 256 MB of memory installed and cannot be
 # allocated to this job. The 'diagnose' command can be used with
 # various flags to obtain detailed information about jobs, nodes,
 # reservations, policies, partitions, etc. The command also
 # performs a number of sanity checks on the data provided and will
 # present warning messages if discrepancies are detected.

 # Let's see if the other single processor jobs cannot run for the
 # same reason.

> diagnose -j | grep Idle | grep " 1 "

 # The grep above selects single processor Idle jobs. The 14th
 # indicates that most single processor jobs currently in the queue
 # require '>=256' MB of RAM, but a few do not. Let's examine job
 # 'fr8n01.1154.0'

> checkjob fr8n01.1154.0

 # The rejection reasons for this job indicate that the four idle
 # processors cannot be used due to 'ReserveTime'. This indicates

 # that the processors are idle but that they have a reservation
 # in place that will start before the job being checked could
 # complete. Let's look at one of the nodes.

> checknode fr10n09

 # The output of this command shows that while the node is idle,
 # it has a reservation in place that will start in a little over
 # 23 hours. (All idle jobs which did not require '>=512' MB
 # required over a day to complete.) It looks like there is
 # nothing that can start right now and we will have to live with
 # four idle nodes.

 # Let's look at the reservation which is blocking the start of
 # our single processor jobs.

> showres

 # This command shows all reservations currently on the system.
 # Notice that all running jobs have a reservation in place. Also,
 # there is one reservation for an idle job (Indicated by the 'I'
 # in the 'S', or 'State' column) This is the reservation that is
 # blocking our serial jobs. This reservation was actually created
 # by the backfill scheduler for the highest priority idle job as
 # a way to prevent starvation while lower priority jobs were being
 # backfilled. (The backfill documentation describes the
 # mechanics of the backfill scheduling more fully.)

 # Let's see which nodes are part of the idle job reservation.

> showres -n fr8n01.963.0

 # All of our four idle nodes are included in this reservation.
 # It appears that everything is functioning properly.

 # Let's step further forward in time.

> schedctl -s 100I

> showstats -v

 # We now know that the scheduler is scheduling efficiently. So
 # far, system utilization as reported by 'showstats -v' looks
 # very good. One of the next questions is 'is it scheduling
 # fairly?' This is a very subjective question. Let's look at
 # the user and group stats to see if there are any glaring
 # problems.

> showstats -u

 # Let's pretend we need to now take down the entire system for
 # maintenance on Thursday from 2 to 10 PM. To do this we would
 # create a reservation.

> setres -S

 # Let's shutdown the scheduler and call it a day.

> schedctl -k

Using sample traces
Collecting traces
using Maui
Understanding and manipulating workload traces
Understanding and manipulating resource traces
Running simulation 'sweeps'
The 'stats.sim' file
(Is not erased at the start of each simulation run. It must be manually cleared or moved if statistics are not to be concatenated)
Using the profiler tool
(profiler man page)

16.2 Resource Traces

Resource traces fully describe all scheduling relevant aspects of a batch system's compute resources. In most cases, each resource trace describes a single compute node providing information about configured resources, node location, supported classes and queues, etc. Each resource trace consists of a single line composed of 21 whitespace delimited fields. Each field is described in detail in the table below.

Field Name	Field Index	Data Format	Default Value	Details
Resource Type	1	one of **COMPUTENODE**	**COMPUTENODE**	currently the only legal value is **COMPUTENODE**
Event Type	2	one of **AVAILABLE**, **DEFINED**, or **DRAINED**	[NONE]	when **AVAILABLE**, **DEFINED**, or **DRAINED** is specified, node will start in the state Idle, Down, or Drained respectively. **NOTE**: node state can be modified using the nodectl command.
Event Time	3	<EPOCHTIME>	1	time event occurred. (currently ignored)
Resource ID	4	<STRING>	N/A	for **COMPUTENODE** resources, this should be the name of the node.
Resource Manager Name	5	<STRING>	[NONE]	name of resource manager resource is associated with
Configured Swap	6	<INTEGER>	1	amount of virtual memory (in MB) configured on node
Configured Memory	7	<INTEGER>	1	amount of real memory (in MB) configured on node (i.e. RAM)
Configured Disk	8	<INTEGER>	1	amount of local disk (in MB) on node available to batch jobs
Configured Processors	9	<INTEGER>	1	number of processors configured on node
Resource Frame Location	10	<INTEGER>	1	number of frame containing node (SP2 only)

Resource Slot Location	11	<INTEGER>	1	Number of first frame slot used by node (SP2 only)
Resource Slot Use Count	12	<INTEGER>	1	Number of frame slots used by node (SP2 only)
Node Operating System	13	<STRING>	[NONE]	node operating system
Node Architecture	14	<STRING>	[NONE]	node architecture
Configured Node Features	15	<STRING>	[NONE]	square bracket delimited list of node features/attributes (ie, '[amd][s1200]')
Configured Run Classes	16	<STRING>	[batch:1]	square bracket delimited list of CLASSNAME:CLASSCOUNT pairs.
Configured Network Adapters	17	<STRING>	[NONE]	square bracket delimited list of configured network adapters (ie, '[atm][fddi][ethernet]')
Relative Resource Speed	18	<DOUBLE>	1.0	relative machine speed value
RESERVED FIELD 1	19	<STRING>	[NONE]	[NONE]
RESERVED FIELD 2	20	<STRING>	[NONE]	[NONE]
RESERVED FIELD 3	21	<STRING>	[NONE]	[NONE]

NOTE: if no applicable value is specified, the exact string '**[NONE]**' should be entered.

Sample Resource Trace:

```
 'COMPUTENODE AVAILABLE 0 cluster008 PBS1 423132 256 7140 2 -1
-1 1 LINUX62 AthlonK7 [s950][compute]  [batch:2]
[ethernet][atm] 1.67  [NONE]  [NONE]  [NONE]'
```

16.3 Workload Traces

Workload traces fully describe all scheduling relevant aspects of batch jobs including resources requested and utilized, time of all major scheduling event (i.e., submission time, start time, etc), the job credentials used, and the job execution environment. Each job trace is composed of a single line consisting of 44 whitespace delimited fields as shown in the table below.

16.3.1 Workload Trace Format

Field Name	Field Index	Data Format	Default Value	Details
JobID	1	<STRING>	[NO DEFAULT]	Name of job, must be unique
Nodes Requested	2	<INTEGER>	0	Number of nodes requested (0 = no node request count specified)
Tasks Requested	3	<INTEGER>	1	Number of tasks requested
User Name	4	<STRING>	[NO DEFAULT]	Name of user submitting job
Group Name	5	<STRING>	[NO DEFAULT]	Primary group of user submitting job
Wallclock Limit	6	<INTEGER>	1	Maximum allowed job duration in seconds
Job Completion State	7	<STRING>	Completed	One of Completed, Removed, NotRun
Required Class	8	<STRING>	[DEFAULT:1]	Class/queue required by job specified as square bracket list of <QUEUE>[:<QUEUE INSTANCE>] requirements. (ie, [batch:1])
Submission Time	9	<INTEGER>	0	Epoch time when job was submitted
Dispatch Time	10	<INTEGER>	0	Epoch time when scheduler requested job begin executing
Start Time	11	<INTEGER>	0	Epoch time when job began executing (NOTE: usually identical to 'Dispatch Time')
Completion Time	12	<INTEGER>	0	Epoch time when job completed execution
Required Network Adapter	13	<STRING>	[NONE]	Name of required network adapter if specified
Required Node Architecture	14	<STRING>	[NONE]	Required node architecture if specified
Required Node Operating System	15	<STRING>	[NONE]	Required node operating system if specified
Required Node Memory Comparison	16	one of >, >=, =, <=, <	>=	Comparison for determining compliance with required node memory

Required Node Memory	17	<INTEGER>	0	Amount of required configured RAM (in MB) on each node
Required Node Disk Comparison	18	one of >, >=, =, <=, <	>=	Comparison for determining compliance with required node disk
Required Node Disk	19	<INTEGER>	0	Amount of required configured local disk (in MB) on each node
Required Node Attributes	20	<STRING>	[NONE]	square bracket enclosed list of node features required by job if specified (ie '[fast][ethernet]')
System Queue Time	21	<INTEGER>	0	Epoch time when job met all fairness policies
Tasks Allocated	22	<INTEGER>	<TASKS REQUESTED>	Number of tasks actually allocated to job (NOTE: in most cases, this field is identical to field #3, Tasks Requested)
Required Tasks Per Node	23	<INTEGER>	-1	Number of Tasks Per Node required by job or '-1' if no requirement specified
QOS	24	<STRING>[:<STRING>]	[NONE]	QOS requested/delivered using the format <QOS_REQUESTED>[:<QOS_DELIVERED>] (ie, 'hipriority:bottomfeeder')
JobFlags	25	<STRING>[:<STRING>]...	[NONE]	square bracket delimited list of job attributes (i.e., [BACKFILL][BENCHMARK][PREEMPTEE])
Account Name	26	<STRING>	[NONE]	Name of account associated with job if specified
Executable	27	<STRING>	[NONE]	Name of job executable if specified
Comment	28	<STRING>	[NONE]	Resource manager specific list of job attributes if specified. See the Resource Manager Extension Overview for more info.
Bypass Count	29	<INTEGER>	-1	Number of time job was bypassed by lower priority jobs via backfill or '-1' if not specified
ProcSeconds Utilized	30	<DOUBLE>	0	Number of processor seconds actually utilized by job
Partition Name	31	<STRING>	[DEFAULT]	Name of partition in which job ran
Dedicated Processors per Task	32	<INTEGER>	1	Number of processors required per task
Dedicated Memory per Task	33	<INTEGER>	0	Amount of RAM (in MB) required per task
Dedicated Disk per Task	34	<INTEGER>	0	Amount of local disk (in MB) required per task
Dedicated Swap per Task	35	<INTEGER>	0	Amount of virtual memory (in MB) required per task
Start Date	36	<INTEGER>	0	Epoch time indicating earliest time job can start
End Date	37	<INTEGER>	0	Epoch time indicating latest time by which job must complete
Allocated Host List	38	<STRING>[:<STRING>]...	[NONE]	colon delimited list of hosts allocated to job (i.e., node001:node004) **NOTE**: In Maui 3.0, this field only lists the job's master host.
Resource Manager Name	39	<STRING>	[NONE]	Name of resource manager if specified

Required Host Mask	40	<STRING>[<STRING>]...	[NONE]	List of hosts required by job. (if taskcount > #hosts, scheduler must use these nodes in addition to others, if taskcount < #host, scheduler must select needed hosts from this list)
Reservation	41	<STRING>	[NONE]	Name of reservation required by job if specified
Set Description	42	<STRING>:<STRING>[:<STRING>]	[NONE]	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, (i.e. 'ONEOF:PROCSPEED:350:450:500')
Application Simulator Data	43	<STRING>[:<STRING>]	[NONE]	Name of application simulator module and associated configuration data (i.e., 'HSM:IN=infile.txt:140000;OUT=outfile.txt:500000')
RESERVED FIELD 1	44	<STRING>	[NONE]	RESERVED FOR FUTURE USE

NOTE: if no applicable value is specified, the exact string '**[NONE]**' should be entered.

Sample Workload Trace:

```
 'SP02.2343.0 20  20  570  519  86400  Removed  [batch:1]  887343658  889585185
889585185  889585411  ethernet  R6000  AIX43  >=  256  >=  0  [NONE] 889584538  20  0 0
2  0  test.cmd 1001  6  678.08  0  1  0  0  0  0  0  [NONE]  0  [NONE]  [NONE]  [NONE]
[NONE]  [NONE]'
```

16.3.2 Creating New Workload Traces

Because workload traces and workload statistics utilize the same format, there are trace fields which provide information that is valuable to a statstical analysis of historical system performance but not necessary for the execution of a simulation.

Particularly, in the area of time based fields, there exists an opportunity to overspecify. Which time based fields are important depend on the setting the the JOBSUBMISSIONPOLICY parameter.

JOBSUBMISSIONPOLICY Value	Critical Time Based Fields
NORMAL	WallClock Limit Submission Time StartTime Completion Time
CONSTANTJOBDEPTH **CONSTANTPSDEPTH**	WallClock Limit StartTime Completion Time

NOTE 1: Dispatch Time should always be identical to Start Time
NOTE 2: In all cases, the difference of 'Completion Time - Start Time' is used to determine actual job run time.
NOTE 3: System Queue Time and Proc-Seconds Utilized are only used for statistics gathering purposes and will not alter the behavior of the simulation.
NOTE 4: In all cases, relative time values are important, i.e., Start Time must be greater than or equal to Submission Time and less than Completion Time.

16.4 Simulation Specific Configuration

Under Construction

17.0 Miscellaneous Features

RESDEPTH

17.1 Feedback Script

The 'Feedback Script' facility allows a site 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 is responsible for creating a program capable of processing and acting upon the contents of the command line. The command line arguments passed are a follows

- job name
- user name
- user email
- final job state
- QOS requested
- epoch time job was submitted
- epoch time job started
- epoch time job completed
- job XFactor
- job wallclock limit
- processors requested
- memory requested
- average per task cpu load
- maximum per task cpu load
- average per task memory usage
- maximum per task memory usage

For many sites, the feedback script is useful as a means of letting user's know that accuracy of their wallclock limit estimate, as well as the cpu efficiency, and memory usage pattern of their job. The feedback script may 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 (ie. "Dear John, you submitted job X requesting 128MB of memory per task. It actually utilized 253 MB of memory per task potentially wreaking havoc with the entire system! Please improve your resource usage estimates in future jobs!")
- update system databases
- take system actions based on job completion statistics

NOTE: some of these fields may be set to zero if the underlying OS/Resource Manager does not support the necessary data collection.

Appendix A Case Studies

A.1 Case Study: Mixed Parallel/Serial Homogeneous Cluster

Overview:

A multi-user site wishes to control the distribution of compute cycles while minimizing job turnaround time and maximizing overall system utilization.

Resources:

Compute Nodes: 64 2 way SMP Linux based nodes, each with 512 MB of RAM and 16 GB local scratch space
Resource Manager: OpenPBS 2.3
Network: 100 MB switched ethernet

Workload:

Job Size: range in size from 1 to 32 processors with approximately the following quartile job frequency distribution
1 - 2, 3 - 8, 9 - 24, and 25 - 32 nodes.

Job Length: jobs range in length from 1 to 24 hours

Job Owners: job are submitted from 6 major groups consisting of a total of about 50 users

NOTES: During prime time hours, the majority of jobs submitted are smaller, short running development jobs where users are testing out new code and new data sets. The owners of these jobs are often unable to proceed with their work until a job they have submitted completes. Many of these jobs are interactive in nature. Throughout the day, large, longer running production workload is also submitted but these jobs do not have comparable turnaround time pressure.

Constraints: (Must do)

The groups 'Meteorology' and 'Statistics' should receive approximately 45 and 35% of the total delivered cycles respectively. Nodes cannot be shared amongst tasks from different jobs.

Goals: (Should do)

The system should attempt to minimize turnaround time during primetime hours (Mon - Fri, 8:00 AM to 5:00 PM) and maximize system utilization during all other times. System

maintenance should be efficiently scheduled around

Analysis:

The network topology is flat and and nodes are homogeneous. This makes life significantly simpler. The focus for this site is controlling distribution of compute cycles without negatively impacting overall system turnaround and utilization. Currently, the best mechanism for doing this is Fairshare. This feature can be used to adjust the priority of jobs to favor/disfavor jobs based on fairshare targets and historical usage. In essence, this feature improves the turnaround time of the jobs not meeting their fairshare target at the expense of those that are. Depending on the criticality of the delivered cycle distribution constraints, this site might also wish to consider an allocations bank such as PNNL's QBank which enables more stringent control over the amount of resources which can be delivered to various users.

To manage the primetime job turnaround constraints, a standing reservation would probably be the best approach. A standing reservation can be used to set aside a subset of the nodes for quick turnaround jobs. This reservation can be configured with a time based access point to allow only jobs which will complete within some time X to utilize these resources. The reservation has advantages over a typical queue based solution in this case in that these quick turnaround jobs can be run anywhere resources are available, either inside, or outside the reservation, or even crossing reservation boundaries. The site does not have any hard constraints about what is acceptable turnaround time so the best approach would probably be to analyze the site's workload under a number of configurations using the simulator and observe the corresponding scheduling behavior.

For general optimization, there are a number of scheduling aspects to consider, scheduling algorithm, reservation policies, node allocation policies, and job prioritization. It is almost always a good idea to utilize the scheduler's backfill capability since this has a tendency to increase average system utilization and decrease average turnaround time in a surprisingly fair manner. It does tend to favor somewhat small and short jobs over others which is exactly what this site desires. Reservation policies are often best left alone unless rare starvation issues arise or quality of service policies are desired. Node allocation policies are effectively meaningless since the system is homogeneous. The final scheduling aspect, job prioritization, can play a significant role in meeting site goals. To maximize overall system utilization, maintaining a significant Resource priority factor will favor large resource (processor) jobs, pushing them to the front of the queue. Large jobs, though often only a small portion of a site's job count, regularly account for the majority of a site's delivered compute cycles. To minimize job turnaround, the XFactor priority factor will favor short running jobs. Finally, in order for fairshare to be effective, a significant Fairshare priority factor must be included.

Configuration:

For this scenario, a resource manager configuration consisting of a single, global queue/class with no constraints would allow Maui the maximum flexibility and opportunities

for optimization.

The following Maui configuration would be a good initial stab.

maui.cfg

```
# reserve 16 processors during primetime for jobs requiring
less than 2 hours to complete

SRNAME[0]        fast
SRTASKCOUNT[0]   16
SRDAYS[0]        MON TUE WED THU FRI
SRSTARTTIME[0]   8:00:00
SRENDTIME[0]     17:00:00
SRMAXTIME[0]     2:00:00

# prioritize jobs for Fairshare, XFactor, and Resources

RESOURCEWEIGHT   20
XFACTORWEIGHT    100
FAIRSHAREWEIGHT  100

# disable SMP node sharing

NODEACCESSPOLICY  DEDICATED
```
---

fs.cfg

```
Group:Meterology  FSTARGET=45
Group:Statistics  FSTARGET=35
```

Monitoring:

The command 'diagnose -f' will allow you to monitor the effectiveness of the fairshare component of your job prioritization. Adjusting the Fairshare priority factor up/or down will make fairshare more/less effective. Note that a tradeoff must occur between fairshare and other goals managed via job prioritization. 'diagnose -p' will help you analyze the priority distributions of the currently idle jobs. The 'showgrid AVGXFACTOR' command will provide a good indication of average job turnaround while the 'profiler' command will give an excellent analysis of longer term historical performance statistics.

Conclusions:

Any priority configuration will need to be tuned over time because the effect of priority weights is highly dependent upon the site specific workload. Additionally, the priority weights themselves are part of a feedback loop which adjust the site workload. However,

most sites quickly stabilize and significant priority tuning is unnecessary after a few days.

A.2 Case Study: Semi-Partitioned Heterogeneous Cluster Dedicated to Parallel and Time-sharing Serial Jobs

Overview:

A site possessing a mixture of uniprocessor and dual processor nodes desires to dedicate a subset of nodes to time-sharing serial jobs, a subset to parallel batch jobs, and provide a set of nodes to be used as 'overflow'.

Resources:

Compute Nodes: Group A: 16 uniprocessor Linux based nodes, each with 128 MB of RAM and 1 GB local scratch space
Group B: 8 2 way SMP Linux based nodes, each with 256 MB of RAM and 4 GB local scratch space
Group C: 8 uniprocessor Linux based nodes, each with 192 MB of RAM and 2 GB local scratch space

Resource Manager: OpenPBS 2.3
Network: 100 MB switched ethernet

Workload:

Job Size: range in size from 1 to 32 processors with approximately the following quartile job frequency distribution
1 - 2, 3 - 8, 9 - 24, and 25 - 32 nodes.

Job Length: jobs range in length from 1 to 24 hours

Job Owners: job are submitted from 6 major groups consisting of a total of about 50 users

NOTES: During prime time hours, the majority of jobs submitted are smaller, short running development jobs where users are testing out new code and new data sets. The owners of these jobs are often unable to proceed with their work until a job they have submitted completes. Many of these jobs are interactive in nature. Throughout the day, large, longer running production workload is also submitted but these jobs do not have comparable turnaround time pressure.

Constraints: (Must do)

Nodes in Group A must run only parallel jobs. Nodes in Group B must only run serial jobs, with up to 4 serial jobs per node. Nodes in Group C must not be used unless a job cannot locate resources elsewhere.

Goals: (Should do)

The scheduler should attempt to intelligently load balance the timesharing nodes.

Analysis:

As in Case Study 1, The network topology is flat and and nodes are homogeneous within each group. The only tricky part of this configuration is the 'overflow' group. The easiest configuration is to create two PBS queues, **serial** and **parallel**, with appropriate min and max node counts as desired. By default, Maui interprets the PBS 'exclusive hostlist' queue attribute as constraining jobs in the queue to run only on the nodes contained in the hostlist. We can take advantage of this behavior to assign nodes in Group A and Group C to the queue 'parallel' while the nodes in Group B and Group C are assigned to the queue 'serial' (The same can be done with classes if using Loadleveler) Maui will incorporate this queue information when making scheduling decisions.

The next step is to make the scheduler use the 'overflow' nodes of group C only as a last resort. This can be accomplished using a negative affinity standing reservation. This configuration will tell the scheduler that these nodes can be used, but should only be used if it cannot find compute resources elsewhere.

The final step, load balancing, is accomplished in two parts. First, the nodes in group B must be configured to allow up to 4 serial jobs to run at a time. This is best accomplished using the PBS 'virtual nodes' feature. To load balance, simply select the CPULOAD allocation algorithm in Maui. This algorithm will instruct Maui to schedule the job based on which node has the most available, unused idle CPU time.

Configuration:

This site requires both resource manager and scheduler configuration.

The following Maui configuration would be needed.

maui.cfg

```
# reserve 'overflow' processors

SRNAME[0]       overflow
SRHOSTLIST[0]   cn0[1-8]           # hostname regular
expression
SRCLASSLIST[0]  parallel-  batch-  # use minus sign to
indicate negative affinity
```

```
ALLOCATIONPOLICY  CPULOAD

# allow SMP node sharing

NODEACCESSPOLICY  SHARED
```

PBS qmgr config

```
set queue serial resources_max.nodeccount=1
set queue serial acl_hosts=an01+an02+...an16+cn01+cn02+...cn08
set queue serial acl_host_enable=true

set queue parallel resources_min.nodecount=2
set queue parallel
acl_hosts=bn01+bn02+...bn08+cn01+cn02+...cn08
set queue parallel acl_host_enable=true
```

PBS 'server_priv/nodes' file

```
bn01  np=4
bn01  np=4
...
```

Monitoring:

Conclusions:

A.3 Case Study: Development O2K

Overview:

A 64 proc O2K system needs to be scheduled with a significant 'background' load.

Resources:

Compute Nodes: 64 processor, 32 GB O2K system
Resource Manager: OpenPBS 2.3
Network: InternalSGI network

Workload:

Job Size: range in size from 1 to 32 processors.

Job Length: jobs range in length from 15 minutes to 48 hours.

Job Owners: various

NOTES: This is a login/development machine meaning at any given time, there may be a significant load originating with jobs/processes outside of the resource manager's view or control. The major scheduling relevant impact of this is in the area of cpu load and real memory consumption.

Constraints: (Must do)

The scheduler must run the machine at maximum capacity without overcommitting either memory or processors. A significant and variable background load exists from jobs submitted outside of the resource manager's view or control. The scheduler must track and account for this load and allow space for some variability and growth of this load over time. The scheduler should also 'kill' any job which violates its requested resource allocation and notify the associated user of this violation.

Goals: (Should do)

The scheduler should maximize the throughput associated with the queued jobs while avoiding starvation as a secondary concern.

Analysis:

The background load causes many problems in any mixed batch/interactive environment. One problem which will occur results from the fact that a situation may arise in which the highest priority batch job cannot run. Maui can make a reservation for this highest priority job but because their are no constraints on the background load, Maui cannot determine when this background load will drop enough to allow this job to run. By default, it optimistically attempts a reservation for the next scheduling iteration, perhaps 1 minute out. The problem is

that this reservation now exists one minute out and when Maui attempts to backfill, it can only consider jobs which request less than one minute or which can fit 'beside' this high priority job. The next iteration, Maui still cannot run the job because the background load has not dropped and again creates a new reservation for one minute out.

The background load has basically turned batch scheduling into an exercise in 'resource scavenging'. If the priority job reservation were not there, other smaller queued jobs might be able to run. Thus to maximize the 'scavenging' effect, the scheduler should be configured to allow this high priority job 'first dibs' on all available resources but prevent it from reserving these resources if it cannot run immediately.

Configuration:

The configuration needs to accomplish several main objectives including:

- track the background load to prevent oversubscription
- favor small, short jobs to maximize job turnaround
- prevent blocked high priority jobs from creating reservations
- interface to an allocation manager to charge for all resource usage based on utilized CPU time
- cancel jobs which exceed specified resource limits
- notify users of job cancellation due to resource utilization limit violations

The following Maui config file should work.

maui.cfg

```
# allow jobs to share node
NODEACCESSPOLICY  SHARED

# track background load
NODELOADPOLICY            ADJUSTPROCS
NODEUNTRACKEDLOADFACTOR   1.2

# favor short jobs, disfavor large jobs
QUEUETIMEWEIGHT   0
RESOURCEWEIGHT    -10
PROCWEIGHT        128
MEMWEIGHT         1
XFACTOR           1000

# disable priority reservations for the default QOS
QOSFLAGS[0]       NORESERVATION

# debit by CPU
BANKTYPE          QBANK
BANKSERVER        develop1
BANKPORT          2334
BANKCHARGEMODE    DEBITSUCCESSFULLCPU
```

```
# kill resource hogs
RESOURCEUTILIZATIONPOLICY ALWAYS
RESOURCEUTILIZATIONACTION CANCEL

# notify user of job events

NOTIFYSCRIPT  tools/notify.pl
```

Monitoring:

The most difficult aspects of this environment are properly 'reserving' space for the untracked 'background' load. Since this load is outside the viewing/control of the scheduler/resource manager, there are no constraints on what it can do. It could instant grow and overwhelm the machine, or just as easily disappear. The parameter 'NODEUNTRACKEDLOADFACTOR' provides 'slack' for this background load to grow and shrink. However, since there is now control over the load, the effectiveness of this parameter will depend on the statistical behavior of this load. The greater the value, the more slack provided, the less likely the system is to be overcommitted; however, a larger value also means more resources are in this 'reserve' which are unavailable for scheduling. The right solution is to migrate the users over to the batch system or provide them with a constrained resource 'box' to play in, either through a processor partition, another system, or via a logical software system. The value in the 'box' is that it prevents this unpredictable background load from wreaking havoc with an otherwise sane dedicated resource reservation system. Maui can reserve resource for jobs according to all info currently available. However the unpredictable nature of the background load may mean those resources are not available when they should be resulting in cancelled reservations and the inability to enforce site policies and priorities.

The second aspect of this environment which must be monitored is the trade-off between high job throughput and job starvation. The 'locally greedy' approach of favoring the smallest, shortest jobs will have a negative effect on larger and longer jobs. The large, long jobs which have been queued for some time can be pushed to the front of the queue by increasing the QUEUETIMEWEIGHT factor until a satisfactory balance is achieved.

Conclusions:

Mixed batch/non-batch systems are very, very nasty. :)

A.4 Case Study: Standard Production SP (Under Construction)

Overview:

An 8 node, 32 processor heterogeneous SP2 system is to be scheduled in a shared node manner.

Resources:

Compute Nodes: 8 node, 32 processor, 24 GB SP2 system
Resource Manager: Loadleveler
Network: IBM High Performance Switch (essentially All-to-All connected)

Workload:

Job Size: range in size from 1 to 16 processors.

Job Length: jobs range in length from 15 minutes to 48 hours.

Job Owners: various

Constraints: (Must do)

Goals: (Should do)

Analysis:

Configuration:

Monitoring:

Conclusions:

A.5 Case Study: Multi-Queue Cluster with QOS and Charge Rates

Overview:

A 160 node, uniprocessor Linux cluster is to be used to support various organizations within an enterprise. The ability to receive improved job turnaround time in exchange for a higher charge rate is required. A portion of the system must be reserved for small development jobs at all times.

Resources:

Compute Nodes: 128 800 MHz uniprocessor nodes w/512 MB each, running Linux 2.4
32 1.2 GHz uniprocessor nodes w/2 GB each, running Linux 2.4

Resource Manager: OpenPBS 2.3
Network: 100 MB ethernet

Workload:

Job Size: range in size from 1 to 80 processors.

Job Length: jobs range in length from 15 minutes to 24 hours.

Job Owners: various

Constraints: (Must do)

The management desires the following queue structure:

QueueName	Nodes	MaxWallTime	Priority	ChargeRate
Test	<=16	00:30:00	100	1x
Serial	1	2:00:00	10	1x
Serial-Long	1	24:00:00	10	2x
Short	2-16	4:00:00	10	1x
Short-Long	2-16	24:00:00	10	2x
Med	17-64	8:00:00	20	1x
Med-Long	17-64	24:00:00	20	2x
Large	65-80	24:00:00	50	2x
LargeMem	1	8:00:00	10	4x

For charging, management has decided to charge by job walltime since the nodes will not be shared. Management has also dictated that 16 of the uniprocessor nodes should be dedicated to running small jobs requiring 16 or fewer nodes. Management has also decided that it would like to allow only serial jobs to run on the large memory nodes and would like to charge these jobs at a rate of 4x. There are no constraints on the remaining nodes.

Goals: (Should do)

This site has goals which are focused more on a supplying a straightforward queue environment to the end users than on maximizing the scheduling performance of the system. The Maui configuration has the primary purpose of faithfully reproducing the queue constraints above while maintaining reasonable scheduling performance in the process.

Analysis:

Since we are using PBS as the resource manager, this should be a pretty straightforward process. It will involve setting up an allocations manager (to handle charging), configuring queue priorities, and creating a system reservation to manage the 16 processors dedicated to small jobs, and another for managing the large memory nodes.

Configuration:

This site has a lot going on. There will be several aspects of configuration, however, they are not too difficult individually.

First, the queue structure. The best place to handle this is via the PBS configuration. Fire up 'qmgr' and set up the nine queues described above. PBS supports the node and walltime constraints as well as the queue priorities. (Maui will pick up and honor queue priorities configured within PBS. Alternatively, you can also specify these priorities directly within the Maui 'fs.cfg' file for resource managers which do not support this capability.) We will be using QBank to handle all allocations and so, will want to configure the the 'per class charge rates' there. (Note: QBank 2.9 or higher is required for per class charge rate support.)

Now, two reservations are needed. The first reservation will be for the 16 small memory nodes. It should only allow node access to jobs requesting up to 16 processors. In this environment, this is probably most easily accomplished with a reservation class ACL containing the queues which allow 1 - 16 node jobs.

Monitoring:

Conclusions:

Appendix B: Extension Interface

Maui supports an extension interface which allows external libraries to be linked to the Maui server. This interface provides these libraries with full access to and control over all Maui objects and data. It also allows this library the ability to use or override most major Maui functions.

The purpose of this library is to allow the development and use of extension modules, or plug-ins, similar to those available for web browsers. One such library, G2, currently extends many core Maui capabilities in the areas of resource management, resource allocation, and scheduling.

Appendix C: Adding New Algorithms with the 'Local' Interface

(Under Construction)

Appendix D: Adjusting Defaulting Limits

Maui 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, it still contains a number of archaic static structures defined in header files. These structures limit the default number of jobs, reservations, nodes, etc, which Maui can handle and are set to values which provide a reasonable compromise between capability and memory consumption for most sites. However, many sites desire to increase some of these settings to extend functionality, or decrease them to save consumed memory. The most common parameters are listed below and can be adjusted by simply modifying the appropriate **#define** and rebuilding Maui.

Parameter	Location	Default	Max Tested	Description
MAX_MATTR (MAX_ATTR in Maui 3.0.7 and earlier)	maui.h	128	512	total number of distinct node attributes (PBS node attributes/LL node features) Maui can track
MAX_MCLASS	msched-common.h	16	64	total number of distinct job classes/queues available
MAX_MJOB	maui.h	4096 (512 in Maui 3.0.5 and earlier)	8192 (4096 in Maui 3.0.5 and earlier)	maximum total number of idle/active jobs Maui can see and process
MAX_MNODE (MAX_NODE in Maui 3.0.6 and earlier)	msched-common.h	1560 (1032 in Maui 3.0.6 and earlier)	8192	maximum number of compute nodes Maui can see and process

MAX_MPARTITION (MAX_PARTITION in Maui 3.0 and earlier)	maui.h	4	16	maximum number of partitions supported
MAX_MQOS	maui.h	128	128	total number of distinct QOS objects available to jobs
MAX_MRES_DEPTH	maui.h	256	256	total number of distinct reservations allowed per node
MAX_SRESERVATION	maui.h	128	256	total number of distinct standing reservations available
MAX_MTASK (MAX_TASK in Maui 3.2.0 and earlier)	msched-common.h	1560	10000	total number of tasks allowed per job

Maui 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 'Local.c' module incorporates the algorithm of interest into the main code. The following scheduling areas are currently handled via the 'Local.c' hooks.

Local Job Attributes
Local Node Allocation Policies
Local Job Priorities
Local Fairness Policies

Overview of Major Maui Structures (Under Construction)

Nodes
mnode_t

Jobs
mjob_t

Reservations
mres_t

Partitions
mpart_t

QOS
mqos_t

Appendix E: Security

E.1 Security Configuration

Maui 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 which are trusted and have access to privileged commands and data.

With regards to users, Maui breaks access into three distinct levels.

E.1.1 Level 1 Maui Admin (Administrator Access)

A level 1 Maui Admin has global access to information and unlimited control over scheduling operations. He is allowed to control scheduler configuration, policies, jobs, reservations, and all scheduling functions. He is also granted access to all available statistics and state information. Level 1 admins are specified using the ADMIN1 parameter.

E.1.2 Level 2 Maui Admin (Operator Access)

Level 2 Maui Admins are specified using the ADMIN2 parameter. The users listed under this parameter are allowed to change all job attributes and are granted access to all informational Maui commands.

E.1.3 Level 3 Maui Admin (Help Desk Access)

Level 3 administrators users a specified via the ADMIN3 parameter. They are allowed access to all informational Maui commands. They cannot change scheduler or job attributes.

E.1.4 Admininstrative Hosts

If specified, the ADMINHOST parameter allows a site to specify a subset of trusted hosts. All administrative commands (level 1-3) will be rejected unless they are received from one of the hosts listed.

E.2 Interface Security

As part of the U.S Department of Energy SSS Initiative, Maui interface security is being enhanced to allow full encryption of data and GSI-like security.

If these mechanisms are not enabled, Maui also provides a simple 'secret checksum' based security model. Under this model, each client request is packaged with the client ID, a timestamp, and a checksum of the entire request generated using a secret site

selected key (checksum seed). This key is selected when the Maui configure script is run and may be regenerated at any time by rerunning configure and rebuilding Maui.

E.2.1 Interface Development Notes

Details about the checksum generation algorithm can be found in the Socket Protocol Description document.

Appendix F: Maui Parameters

See the Parameters Overview in the Maui Admin Manual for further information about specifying parameters.

Name	Format	Default Value	Description	Example
ACCOUNTCFG[<ACCOUNTID>]	list of zero or more space delimited <ATTR>=<VALUE> pairs where <ATTR> is one of the following: **PRIORITY**, **FSTARGET**, **QLIST**, **QDEF**, **PLIST**, **PDEF**, **FLAGS**, or a fairness policy specification.	[NONE]	specifies account specific attributes. See the flag overview for a description of legal flag values. **NOTE:** Only available in Maui 3.0.7 and higher.	`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.)
ACCOUNTFSWEIGHT	<INTEGER>	0	specifies the priority weight to be applied to the account fairshare factor. (See Fairshare Priority Factor)	`ACCOUNTFSWEIGHT 10`
ACCOUNTWEIGHT	<INTEGER>	0	specifies the priority weight to be applied to the specified account priority. (See Credential Priority Factor)	`ACCOUNTWEIGHT 100`
ADMIN1	space delimited list of user names	root	users listed under the parameter ADMIN1 are allowed to perform any scheduling function. They have full control over the scheduler and access to all data. The first user listed in the ADMIN1 user list is considered to be the 'primary admin' and is the ID under which maui must be started and run. Valid values include user names or the keyword 'ALL'.	`ADMIN1 mauiuser steve scott jenny` (all users listed have full access to Maui control commands and maui data. Maui must be started by and run under the 'mauiuser' user id since mauiuser is the primary admin.
ADMIN2	space delimited list of user names	[NONE]	users listed under the parameter ADMIN2 are allowed to change all job attributes and are granted access to all informational Maui commands. Valid values include user names or the keyword 'ALL'.	`ADMIN2 jack karen` (jack and karen can modify jobs, i.e., 'canceljob, setqos, setspri, etc.) and can run all Maui information commands).
ADMIN3	space delimited list of user names	[NONE]	users listed under the parameter ADMIN3 are allowed access to all informational maui commands. They cannot change scheduler or job attributes. Valid values include user names or the keyword 'ALL'.	`ADMIN3 ops` (user `ops` can run all informational command such as 'checkjob' or checknode')

BACKFILLDEPTH	<INTEGER>	0 (no limit)	specifies the number idle jobs to evaluate for backfill. The backfill algorithm will evaluate the top <X> priority jobs for scheduling. By default, all jobs are evaluated.	`BACKFILLDEPTH 128` (evaluate only the top 128 highest priority idle jobs for consideration for backfill)
BACKFILLMETRIC	one of the following **PROCS**, **PROCSECONDS**, **SECONDS**, **PE**, or **PESECONDS**	**PROCS**	specifies the criteria used by the backfill algorithm to determine the 'best' jobs to backfill. Only applicable when using BESTFIT or GREEDY backfill algorithms	`BACKFILLMETRIC PROCSECONDS`
BACKFILLPOLICY	one of the following: **FIRSTFIT**, **BESTFIT**, **GREEDY**, or **NONE**	**FIRSTFIT**	specifies what backfill algorithm will be used	`BACKFILLPOLICY BESTFIT`
BANKCHARGEPOLICY	one of **DEBITALLWC**, **DEBITSUCCESSFULWC**, **DEBITSUCCESSFULCPU**, or **DEBITSUCCESSFULPE**	**DEBITSUCCESSFULWC**	specifies how Maui should 'charge' a completed job against an allocation manager. See the Allocation Manager Overview for details.	`BANKCHARGEPOLICY DEBITALLWC` (Maui will charge an account for the resources dedicated to a job regardless of how well the job uses these resources and regardless of whether or not the job completes successfully.)
BANKDEFERJOBONFAILURE	ON or OFF	OFF	specifies whether or not Maui should defer jobs if the allocation bank is failing	`BANKDEFERJOBONFAILURE ON`
BANKFALLBACKACCOUNT	<STRING>	[NONE]	account to use if specified account is out of allocations	`BANKFALLBACKACCOUNT bottomfeeder`
BANKPORT	<INTEGER>	40560	port to use to contact allocation manager bank	`BANKPORT 40555`
BANKSERVER	<STRING>.	[NONE]	name of host on which allocation manager bank service resides	`BANKSERVER zephyr1`
BANKTIMEOUT	<INTEGER>	9	number of seconds Maui will wait before timing out on a bank connection	`BANKTIMEOUT 00:00:30`
BANKTYPE	one of **QBANK**, **RESD**, or **FILE**	[NONE]	specifies the type of allocation bank to use	`BANKTYPE QBANK`
BYPASSWEIGHT	<INTEGER>	0	specifies the weight to be applied to a job's backfill bypass count when determining a job's priority	`BYPASSWEIGHT 5000`
CHECKPOINTEXPIRATIONTIME	[[[DD:]HH:]MM:]SS	INFINITY	specifies how 'stale' checkpoint data can be before it is ignored and purged.	`CHECKPOINTEXPIRATIONTIME 1:00:00:00` (Expire checkpoint data which has been stale for over one day)
CHECKPOINTFILE	<STRING>	maui.ck	name (absolute or relative) of the Maui checkpoint file.	`CHECKPOINTFILE /var/adm/maui/maui.ck` (Maintain the Maui checkpoint file in the file specified)
CHECKPOINTINTERVAL	[[[DD:]HH:]MM:]SS	00:05:00	time between automatic Maui checkpoints	`CHECKPOINTINTERVAL 00:15:00` (Maui should checkpoint state information every 15 minutes)

CLASSCFG[<CLASSID>]	list of zero or more space delimited <ATTR>=<VALUE> pairs where <ATTR> is one of the following: **PRIORITY**, **FSTARGET**, **QLIST**, **QDEF**, **PLIST**, **PDEF**, **FLAGS**, or a fairness policy specification.	[NONE]	specifies class specific attributes. See the flag overview for a description of legal flag values. **NOTE:** Only available in Maui 3.0.7 and higher.	`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	<INTEGER>	0	specifies the weight to be applied to the class priority of each job (See Cred Factor)	`CLASSWEIGHT 10`
CLIENTKEY[<X>]	<INTEGER>	[NONE]	specifies the session key which Maui will use to communicate with the named peer daemon. **NOTE:** this parameter may only be specified in the **maui-private.cfg** config file)	`CLIENTKEY[silverB] 0x3325584` (Maui will use the session key `0x3325584` for encrypting and decrypting messages communicated from `silverB`)
CLIENTTIMEOUT	[[[DD:]HH:]MM:]SS	00:00:30	time which Maui client commands will wait for a response from the Maui server (NOTE: may also be specified as an environment variable)	`CLIENTTIMEOUT 00:15:00` (Maui clients will wait up to 15 minutes for a response from the server before timing out)
CREDWEIGHT	<INTEGER>	1	specifies the credential component weight (See Cred Factor) NOTE: this parameter was named DIRECTSPECWEIGHT prior to Maui 3.0.7.	`CREDWEIGHT 2`
DEFAULTCLASSLIST	space delimited list of one or more <STRING>'s	[NONE]	specifies the default classes supported on each node for RM systems which do not provide this information	`DEFAULTCLASSLIST serial parallel`
DEFERCOUNT	<INTEGER>	24	specifies the number of times a job can be deferred before it will be placed in batch hold.	`DEFERCOUNT 12`
DEFERSTARTCOUNT	<INTEGER>	1	specifies number of time a job will be allowed to fail in its start attempts before being deferred.	`DEFERSTARTCOUNT 3`
DEFERTIME	[[[DD:]HH:]MM:]SS	1:00:00	specifies amount of time a job will be held in the deferred state before being released back to the Idle job queue	`DEFERSTARTTIME 0:05:00`
DIRECTSPECWEIGHT	<INTEGER>	0	specifies the credential component weight (See Cred Factor) NOTE: this parameter has been renamed CREDWEIGHT in Maui 3.0.7 and higher.	`DIRECTSPECWEIGHT 2`

DISKWEIGHT	<INTEGER>	0	specifies the priority weight to be applied to the amount of dedicated disk space required per task by a job (in MB)	`RESWEIGHT 10`
				`DISKWEIGHT 100`
				(if a job requires 12 tasks and 512 MB per task of dedicated local disk space, Maui will increase the job's priority by 10 * 100 * 12 * 512)

DISPLAYFLAGS	one or more of the following values (space delimited) NODECENTRIC	[NONE]	specifies flags which control how maui client commands will display various information	`DISPLAYFLAGS NODECENTRIC`
DOWNNODEDELAYTIME*	[[[DD:]HH:]MM:]SS	24:00:00	default time an unavailable node (Down or Drain) is marked unavailable	`DOWNNODEDELAYTIME 1:00:00` (Maui will assume 'down' nodes will be available 1 hour after they go down unless a system reservation is placed on the node)
ENFORCERESOURCELIMITS	one of the following: ON or OFF	OFF	specifies whether or not Maui should take action when a job exceeds its requested resource usage.	`ENFORCERESOURCELIMITS ON`
FEATURENODETYPEHEADER	<STRING>	[NONE]	specifies the header used to specify node type via node features (ie, LL features or PBS node attributes).	`FEATURENODETYPEHEADER xnt` (Maui will interpret all node features with the leading string `xnt` as a nodetype specification - as used by QBank and other allocation managers, and assign the associated value to the node. i.e., xntFast)
FEATUREPARTITIONHEADER	<STRING>	[NONE]	specifies the header used to specify node partition via node features (ie, LL features or PBS node attributes).	`FEATUREPARTITIONHEADER xpt` (Maui will interpret all node features with the leading string `xpt` as a partition specification and assign the associated value to the node. i.e., xptGold)
FEATUREPROCSPEEDHEADER	<STRING>	[NONE]	specifies the header used to extract node processor speed via node features (i.e., LL features or PBS node attributes). NOTE: Adding a trailing '$' character will specifies that only features with a trailing number be interpreted. For example, the header 'sp$' will match 'sp450' but not 'sport'	`FEATUREPROCSPEEDHEADER xps` (Maui 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)
FEEDBACKPROGRAM	<STRING>	[NONE]	specifies the name of the program to be run at the completion of each job. If not fully qualified, Maui will attempt to locate this program in the 'tools' subdirectory.	`FEEDBACKPROGRAM /var/maui/fb.pl` (Maui will run the specified program at the completion of each job.)
FSACCOUNTWEIGHT	<INTEGER>	0	specifies the weight assigned to the account subcomponent of the fairshare component of priority	`FSACCOUNTWEIGHT 10`
FSCAP	<DOUBLE>	0 (NO CAP)	specifies the maximum allowed value for a job's total pre-weighted fairshare component	`FSCAP 10.0` (Maui will not allow a job's pre-weighted fairshare component to exceed 10.0, ie, Priority = FSWEIGHT * MIN(FSCAP,FSFACTOR) + ...)
FSCONFIGFILE	<STRING>	fs.cfg		
FSDECAY	<DOUBLE>	1.0		
FSDEPTH	<INTEGER>	7		
FSGROUPWEIGHT	<INTEGER>	0		`FSGROUPWEIGHT 4`

FSINTERVAL	[[[DD:]HH:]MM:]SS	24:00:00	specifies the length of each fairshare 'window'	`FSINTERVAL 12:00:00` (track fairshare usage in 12 hour blocks)
FSPOLICY	one of the following: **DEDICATEDPS**, **DEDICATEDPES**	[NONE]	specifies the unit of tracking fairshare usage. **DEDICATEDPS** tracks dedicated processor seconds. **DEDICATEDPES** tracks dedicated processor-equivalent seconds	`FSPOLICY DEDICATEDPES` (Maui will track fairshare usage by dedicated process-equivalent seconds)
FSQOSWEIGHT	<INTEGER>	0	specifies the priority weight assigned to the QOS fairshare subcomponent	
FSUSERWEIGHT	<INTEGER>	0	specifies the priority weight assigned to the user fairshare subfactor.	`FSUSERWEIGHT 8`
FSWEIGHT	<INTEGER>	0	specifies the priority weight assigned to the summation of the fairshare subfactors	`FSWEIGHT 500`
GROUPCFG[<GROUPID>]	list of zero or more space delimited <ATTR>=<VALUE> pairs where <ATTR> is one of the following: **PRIORITY**, **FSTARGET**, **QLIST**, **QDEF**, **PLIST**, **PDEF**, **FLAGS**, or a fairness policy specification.	[NONE]	specifies group specific attributes. See the flag overview for a description of legal flag values. **NOTE:** Only available in Maui 3.0.7 and higher.	`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	<INTEGER>	0	specifies the priority weight assigned to the specified group priority (See Direct Spec Factor)	`GROUPWEIGHT 20`
JOBMAXSTARTTIME	[[[DD:]HH:]MM:]SS	-1 (NO LIMIT)	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, the scheduler will cancel the job, believing a system failure has occurred.	`JOBMAXSTARTTIME 2:00:00` (jobs may attempt to start for up to 2 hours before being cancelled by Maui)
JOBMAXOVERRUN	[[[DD:]HH:]MM:]SS	0	amount of time Maui will allow a job to exceed its wallclock limit before it is terminated	`JOBMAXOVERRUN 1:00:00` (allow jobs to exceed their wallclock limit by up to 1 hour)
JOBNODEMATCHPOLICY	zero or more of the following: **EXACTNODE** or **EXACTPROC**	[NONE]	specifies additional constraints on how compute nodes are to be selected. **EXACTNODE** indicates that Maui should select as many nodes as requested even if it could pack multiple tasks onto the same node. **EXACTPROC** indicates that Maui should select only nodes with exactly the number of processors configured as are requested per node even if nodes with excess processors are available.	`JOBNODEMATCHPOLICY EXACTNODE` (In a PBS job with resource specification 'nodes=<x>:ppn=<y>', Maui will allocate exactly <y> task on each of <x> distinct nodes.)

JOBPRIOACCRUALPOLICY	one of the following: **ALWAYS**, **FULLPOLICY**, **QUEUEPOLICY**	**QUEUEPOLICY**	specifies how the dynamic aspects of a job's priority will be adjusted. **ALWAYS** indicates that the job will accrue queuetime based priority from the time it is submitted. **FULLPOLICY** indicates that it will accrue priority only when it meets all queue AND run policies. **QUEUEPOLICY** indicates that it will accrue priority so long as it satisfies various queue policies, i.e. MAXJOBQUEUED.	`JOBPRIOACCRUALPOLICY QUEUEPOLICY` (Maui will adjust the job's dynamic priority subcomponents, i.e., QUEUETIME, XFACTOR, and TARGETQUEUETIME, etc. each iteration that the job satisfies the associated 'QUEUE' policies such as MAXJOBQUEUED.)
JOBSIZEPOLICY	<N/A>	[NONE]	<N/A>	<N/A>
JOBSYNCTIME	[[[DD:]HH:]MM:]:SS	00:10:00	specifies the length of time after which Maui will sync up a job's expected state with an unexpected reported state. IMPORTANT NOTE: Maui will not allow a job to run as long as its expected state does not match the state reported by the resource manager. NOTE: this parameter is named JOBSYNCDEADLINE in Maui 3.0.5 and earlier	`JOBSYNCTIME 00:01:00`
LOGDIR	<STRING>	log	specifies the directory in which log files will be maintained. If specified as a relative path, LOGDIR will be relative to $(MAUIHOMEDIR) (see Logging Overview)	`LOGDIR /tmp` (Maui will record its log files directly into the `/tmp` directory)
LOGFACILITY	colon delimited list of one or more of the following: **fCORE**, **fSCHED**, **fSOCK**, **fUI**, **fLL**, **fSDR**, **fCONFIG**, **fSTAT**, **fSIM**, **fSTRUCT**, **fFS**, **fCKPT**, **fBANK**, **fRM**, **fPBS**, **fWIKI**, **fALL**	**fALL**	specifies which types of events to log (see Logging Overview)	`LOGFACILITY fRM:fPBS` (Maui will log only events involving general resource manager or PBS interface activities.)
LOGFILE	<STRING>	maui.log	name of the maui log file. This file is maintained in the directory pointed to by <LOGDIR> unless <LOGFILE> is an absolute path (see Logging Overview)	`LOGFILE maui.test.log` (Log information will be written to the file `maui.test.log` located in the directory pointed to by the LOGDIR parameter)
LOGFILEMAXSIZE	<INTEGER>	10000000	maximum allowed size (in bytes) the log file before it will be 'rolled' (see Logging Overview)	`LOGFILEMAXSIZE 50000000` (Log files will be rolled when they reach 50 MB in size)

LOGFILEROLLDEPTH	<INTEGER>	2	number of old log files to maintain (i.e., when full, maui.log will be renamed maui.log.1, maui.log.1 will be renamed maui.log.2, ... **NOTE: Only available in Maui 3.0.5 and higher.** (see Logging Overview)	`LOGFILEROLLDEPTH 5` (Maui will maintain the last 5 log files.)
LOGLEVEL	<INTEGER> (0-9)	0	specifies the verbosity of Maui logging where 9 is the most verbose (NOTE: each logging level is approximately an order of magnitude more verbose than the previous level) (see Logging Overview)	`LOGLEVEL 4` (Maui will write all Maui log messages with a threshold of 4 or lower to the 'maui.log' file)
MAXJOBPERUSER	<INTEGER>[,<INTEGER>]	0 (No Limit)	maximum number of active jobs allowed at any given time. See note for Maui 3.0 versions.	
MAXJOBQUEUEDPERUSER	<INTEGER>[,<INTEGER>]	0 (No Limit)	maximum number of idle jobs which can be considered for scheduling and which can acquire 'system queue time' for increasing job priority. See note for Maui 3.0 versions.	
MAXNODEPERUSER	<INTEGER>[,<INTEGER>]	0 (No Limit)	maximum allowed total PE count which can be dedicated at any given time. See note for Maui 3.0 versions.	
MAXPEPERUSER	<INTEGER>[,<INTEGER>]	0 (No Limit)	maximum allowed total PE count which can be dedicated at any given time. See note for Maui 3.0 versions.	
MAXPROCPERUSER	<INTEGER>[,<INTEGER>]	0 (No Limit)	maximum allowed total processors which can be dedicated at any give time. See note for Maui 3.0 versions.	
MAXPSPERUSER	<INTEGER>[,<INTEGER>]	0 (No Limit)	maximum allowed sum of outstanding dedicated processor-second obligations of all active jobs. See note for Maui 3.0 versions.	
MAXWCPERUSER	[[[DD:]HH:]MM:]SS[,[[[DD:]HH:]MM:]SS]	0 (No Limit)	maximum allowed sum of outstanding walltime limits of all active jobs. NOTE: only available in Maui 3.2 and higher.	

MEMWEIGHT[X]	<INTEGER>	0	specifies the coefficient to be multiplied by a job's MEM (dedicated memory in MB) factor	`RESWEIGHT[0] 10`
				`MEMWEIGHT[0] 1000`
				(each job's priority will be increased by 10 * 1000 * its MEM factor)

NODEACCESSPOLICY	one of the following: **DEDICATED**, **SHARED**, or **SINGLEUSER**	**DEDICATED**	specifies whether or not Maui will allow node resources to be shared or dedicated by independent jobs	`NODEACCESSPOLICY SHARED` (Maui will allow resources on a node to be used by more than one job)
NODEALLOCATIONPOLICY	one of the following: **FIRSTAVAILABLE**, **LASTAVAILABLE**, **MINRESOURCE**, **CPULOAD**, **MACHINEPRIO, LOCAL, CONTIGUOUS, MAXBALANCE, or FASTEST**	**LASTAVAILABLE**	specifies how Maui should allocate available resources to jobs. (See the Node Allocation section of the Admin manual for more information)	`NODEALLOCATIONPOLICY MINRESOURCE` (Maui will apply the node allocation policy 'MINRESOURCE' to all jobs by default)
NODECFG[X]	list of space delimited <ATTR>=<VALUE> pairs where <ATTR> is one of the following: **ACCESS, MAXJOB**, **MAXJOBPERUSER**, **MAXLOAD**, **FRAME**, **SLOT**, **SPEED**, **PROCSPEED**, **PARTITION**, **NODETYPE**, **FEATURES**	[NONE]	specifies node-specific attributes for the node indicated in the array field. See the Node Configuration Overview for more information. **NOTE**: this parameter is enabled in Maui 3.0.7 and higher.	`NODECFG[nodeA] MAXJOB=2 SPEED=1.2` (Maui will only only two simultaneous jobs to run on node '`nodeA`' and will assign a relative machine speed of 1.2 to this node.)
NODEDOWNSTATEDELAYTIME	[[[DD:]HH:]MM:]SS	0:00:00	length of time Maui will assume down, drained (offline), or corrupt nodes will remain unavailable for scheduling if a system reservation is not explicitly created for the node. **NOTE**: This parameter is enabled in Maui 3.0.7 and higher.	`NODEDOWNSTATEDELAYTIME 0:30:00` (Maui will assume down, drained, and corrupt nodes are not available for scheduling for at least 30 minutes from the current time. Thus, 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.)
NODELOADPOLICY	one of the following: **ADJUSTSTATE** or **ADJUSTPROCS**	**ADJUSTSTATE**	specifies if a node's load affects its state or its available processors. ADJUSTSTATE tells Maui to mark the node busy when MAXLOAD is reached. ADJUSTPROCS causes the node's available procs to be equivalent to MIN(ConfiguredProcs - DedicatedProcs,MaxLoad - CurrentLoad) NOTE: NODELOADPOLICY only affects a node if MAXLOAD has been set.	`NODELOADPOLICY ADJUSTSTATE` (Maui will mark a node busy if its measured load exceeds its MAXLOAD setting)
NODEMAXLOAD	<DOUBLE>	0.0	specifies that maximum load on a idle of running node. If the node's load reaches or exceeds this value, Maui will mark the node 'busy'	`NODEMAXLOAD 0.75` (Maui will adjust the state of all Idle and Running nodes with a load >= .75 to the state 'Busy')
NODEPOLLFREQUENCY	<INTEGER>	0 (Poll Always)	specifies the number of scheduling iterations between scheduler initiated node manager queries.	`NODEPOLLFREQUENCY 5` (Maui will update node manager based information every 5 scheduling iterations)

NODESETATTRIBUTE	one of **FEATURE**, **NETWORK**, or **PROCSPEED**	[NONE]	specifies the type of node attribute by which node set boundaries will be established. **NOTE**: enabled in Maui 3.0.7 and higher. (See Node Set Overview)	`NODESETATTRIBUTE PROCSPEED` (Maui will create node sets containing nodes with common processor speeds)
NODESETDELAY	[[[DD:]HH:]MM:]SS	0:00:00	specifies the length of time Maui will delay a job if adequate idle resources are available but not adequate resources within node set constraints. **NOTE**: enabled in Maui 3.0.7 and higher. (See Node Set Overview)	`NODESETDELAY 0:00:00` (Maui will create node sets containing nodes with common processor speeds)

NODESETLIST	<STRING>	[NONE]	specifies the list of node attribute values which will be considered for establishing node sets. **NOTE**: enabled in Maui 3.0.7 and higher. (See Node Set Overview)	`NODESETPOLICY ONEOF`
				`NODESETATTRIBUTE FEATURE`
				`NODESETLIST switchA switchB`
				(Maui will allocate nodes to jobs either using only nodes with the 'switchA' feature or using only nodes with the 'switchB' feature.)

NODESETPOLICY	one of **ONEOF**, **FIRSTOF**, or **ANYOF**	[NONE]	specifies how nodes will be allocated to the job from the various node set generated. **NOTE**: enabled in Maui 3.0.7 and higher. (See Node Set Overview)	`NODESETPOLICY ONEOF`
				`NODESETATTRIBUTE NETWORK`
				(Maui will create node sets containing nodes with common network interfaces)

NODESETPRIORITYTYPE	one of **BESTFIT**, **WORSTFIT**, **BESTRESOURCE**, or **MINLOSS**	MINLOSS	specifies how resource sets will be selected when more than one feasible resource can can be found. **NOTE**: This parameter is available in Maui 3.0.7 and higher. (See Node Set Overview)	`NODESETPRIORITYTYPE BESTRESOURCE`
				`NODESETATTRIBUTE PROCSPEED`
				(Maui will select the resource set containing the fastest nodes available)

NODESETTOLERANCE	<FLOAT>	0.0 (Exact match only)	specifies the tolerance for selection of mixed processor speed nodes. A tolerance of **X** allows a range of processors to be selected subject to the constraint
			(Speed.Max - Speed.Min) / Speed.Min <= X
			NOTE: Tolerances are only applicable when NODESETFEATURE is set to PROCSPEED. This parameter is available in Maui 3.0.7 and higher.
			(See Node Set Overview)
				`NODESETATTRIBUTE PROCSPEED`
				`NODESETTOLERANCE 0.5`
				(Maui will only allocate nodes with up to a 50% procspeed difference.)

NODESYNCTIME	[[[DD:]HH:]MM:]SS	00:10:00	specifies the length of time after which Maui will sync up a node's expected state with an unexpected reported state. IMPORTANT NOTE: Maui will not start new jobs on a node with an expected state which does not match the state reported by the resource manager. NOTE: this parameter is named **NODESYNCDEADLINE** in Maui 3.0.5 and earlier.	`NODESYNCTIME 1:00:00`
NODEWEIGHT	<INTEGER>	0	specifies the weight which will be applied to a job's requested node count before this value is added to the job's cumulative priority. **NOTE**: 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. (This will be rectified in future versions.)	`NODEWEIGHT 1000`
NOTIFICATIONPROGRAM	<STRING>	[NONE]	specifies the name of the program to handle all notification call-outs	`NOTIFICATIONPROGRAM tools/notifyme.pl`

PEWEIGHT[X]	<INTEGER>	0	specifies the coefficient to be multiplied by a job's PE (processor equivalent) priority factor	`RESWEIGHT[0] 10`
				`PEWEIGHT[0] 100`
				(each job's priority will be increased by 10 * 100 * its PE factor)

PLOTMAXPROC	<INTEGER>	512	specifies the maximum number of processors requested by jobs to be displayed in matrix outputs (as displayed by the showgrid or profiler commands)	`PLOTMINPROC 1`
				`PLOTMAXPROC 1024`
				(each matrix output will display data in rows for jobs requesting between 1 and 1024 processors)

PLOTMAXTIME	[[[DD:]HH:]MM:]SS	68:00:00	specifies the maximum duration of jobs to be displayed in matrix outputs (as displayed by the showgrid or profiler commands)	`PLOTMINTIME 1:00:00`
				`PLOTMAXTIME 64:00:00`
				(each matrix output will display data in columns for jobs requesting between 1 and 64 hours of run time)

PLOTMINPROC	<INTEGER>	1	specifies the minimum number of processors requested by jobs to be displayed in matrix outputs (as displayed by the showgrid or profiler commands)	`PLOTMINPROC 1`
				`PLOTMAXPROC 1024`
				(each matrix output will display data in rows for jobs requesting between 1 and 1024 processors)

PLOTMINTIME	[[[DD:]HH:]MM:]SS	00:02:00	specifies the minimum duration of jobs to be displayed in matrix outputs (as displayed by the showgrid or profiler commands)	`PLOTMINTIME 1:00:00`
				`PLOTMAXTIME 64:00:00`
				(each matrix output will display data in columns for jobs requesting between 1 and 64 hours of run time)

PLOTPROCSCALE	<INTEGER>	9	specifies the number of rows into which the range of processors requested per job will be divided when displayed in matrix outputs (as displayed by the showgrid or profiler commands)	`PLOTMINPROC 1`
				`PLOTMAXPROC 1024`
				`PLOTPROCSCALE 10`
				(each matrix output will display job data divided into 10 rows which are evenly spaced geometrically covering the range of jobs requesting between 1 and 1024 processors)

PLOTTIMESCALE	<INTEGER>	11	specifies the number of columns into which the range of job durations will be divided when displayed in matrix outputs (as displayed by the showgrid or profiler commands)	`PLOTMINTIME 2:00:00`
				`PLOTMAXTIME 32:00:00`
				`PLOTTIMESCALE 5`
				(each matrix output will display job data divided into 5 columns which are evenly spaced geometrically covering the range of jobs requesting between 2 and 32 hours, i.e., display columns for 2, 4, 8, 16, and 32 hours of walltime)

PREEMPTIONPOLICY	one of the following: **REQUEUE**, **SUSPEND**, **CHECKPOINT**	**REQUEUE**	specifies how preemptible jobs will be preempted (Available in Maui 3.2.2 and higher)	`PREEMPTIONPOLICY  CHECKPOINT` (jobs that are to be preempted will be checkpointed and restarted at a later time)
PROCWEIGHT[X]	<INTEGER>	0	specifies the coefficient to be multiplied by a job's requested processor count priority factor	`PROCWEIGHT  2500`
PURGETIME	[[[DD:]HH:]MM:]SS	0	The amount of time Maui will keep a job or node record for an object no longer reported by the resource manager. Useful when using a resource manager which 'drops' information about a node or job due to internal failures.**NOTE**: In Maui 3.2.0 an higher, this parameter is superseded by **JOBPURGETIME** and **NODEPURGETIME**	`PURGETIME  00:05:00` (Maui will maintain a job or node record for 5 minutes after the last update regarding that object received from the resource manager.)
QOSCFG[<QOSID>]	list of zero or more space delimited <ATTR>=<VALUE> pairs where <ATTR> is one of the following: **PRIORITY**, **FSTARGET**, **QTWEIGHT**, **QTTARGET**, **XFWEIGHT**, **XFTARGET**, **PLIST**, **PDEF**, **FLAGS**, or a fairness policy specification.	[NONE]	specifies QOS specific attributes. See the flag overview for a description of legal flag values. **NOTE:** Available in Maui 3.0.6 and higher. **QOSCFG** supersedes **QOSNAME**, **QOSPRIORITY**, **QOSFLAGS**, and other '**QOS***' parameters.	`QOSCFG[commercial]  PRIORITY=1000 MAXJOB=4  MAXPROCS=80` (Maui 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.)

QOSFEATURES[X]	one or more node feature values or [ANY]	[ANY]	specifies which node features must be present on resources allocated to jobs of the associated QOS. This parameter takes a QOS name as an array index.	`QOSFEATURES[2] wide interactive` (jobs with a QOS value of 2 may only run on nodes with the feature 'wide' AND the feature 'interactive' set)
QOSFLAGS[X]	one or more of the following (space delimited) IGNJOBPERUSER, IGNPROCPERUSER, IGNNODEPERUSER, IGNPSPERUSER, IGNJOBQUEUEDPERUSER, IGNJOBPERGROUP, IGNPROCPERGROUP, IGNPSPERGROUP, IGNJOBQUEUEDPERGROUP, IGNJOBPERACCOUNT, IGNPROCPERACCOUNT, IGNPSPERACCOUNT, IGNJOBQUEUEDPERACCOUNT, IGNSYSMAXPROC, IGNSYSMAXTIME, IGNSYSMAXPS, IGNSRMAXTIME, IGNUSER, IGNGROUP, IGNACCOUNT, IGNSYSTEM, IGNALL, PREEMPT, DEDICATED, RESERVEALWAYS, USERESERVED, NOBF, NORESERVATION, RESTARTPREEMPT	[NONE]	specifies the attributes of the corresponding QOS value See the Admin Manual QOS Overview section for details (NOTE: some flags are only supported under Maui 3.1 and later)	`QOSFLAGS[1] ADVRES IGNMAXJOBPERUSER` (jobs with a QOS value of 1 must run in an advance reservation and can ignore the MAXJOBPERUSER policy)
QOSPRIORITY[X]	<INTEGER>	0	specifies the priority associated with this QOS (NOTE: only used in Maui 3.0.x)	`QOSPRIORITY[2]  1000` (set the priority of QOS 2 to 1000)
QOSQTTARGET[X]	[[[DD:]HH:]MM:]SS	[NONE]	specifies the target job queuetime associated with this QOS	`QOSQTTARGET   2:00:00`
QOSQTWEIGHT[X]	<INTEGER>	0	specifies the 'per QOS' queue time priority weight	`QOSQTWEIGHT    5`

QOSXFTARGET[X]	<DOUBLE>	[NONE]	specifies the expansion factor target used in a job's 'Target Factor' priority calculation	`QOSWEIGHT[3]    10`
				`QOSXFTARGET[3]  5.0`
				(jobs requesting a QOS of 3 will have their priority grow exponentially as the job's minimum expansion factor approaches 5.0)

QOSXFWEIGHT[X]	<INTEGER>	0	specifies the weight which will be added to the base XFWEIGHT for all jobs using QOS 'X'	`XFWEIGHT[0]    100`
				`QOSXFWEIGHT[2] 1000`
				(jobs using QOS '2' will have a XFWEIGHT of 1100 while jobs using other QOS's will have an XFWEIGHT of 100)

QUEUETIMECAP[X]	<DOUBLE>	0 (NO CAP)	specifies the maximum allowed pre-weighted queuetime priority factor.	`QUEUETIMECAP[0]    10000`
				`QUEUETIMEWEIGHT[0] 10`
				(a job that has been queued for 40 minutes will have its queuetime priority factor calculated as 'Priority = QUEUETIMEWEIGHT * MIN(10000,40)')

QUEUETIMEWEIGHT[X]	<INTEGER>	1	specifies multiplier applied to a job's queue time (in minutes) to determine the job's queuetime priority factor	`QUEUETIMEWEIGHT[0] 20` (a job that has been queued for 4:20:00 will have a queuetime priority factor of 20 * 260)
RESCTLPOLICY	one of the following: **ADMINONLY**, **ANY**	**ADMINONLY**	specifies who can create admin reservations (Available in Maui 3.2 and higher)	`RESCTLPOLICY ANY` (any valid user can create an arbitrary admin reservation)
RESDEPTH	<INTEGER>	24	specifies the maximum number of reservations which can be on any single node. IMPORTANT NOTE: on large way SMP systems, this value often must be increased. To be on the safe side, this value should be approximately twice the average sum of admin, standing, and job reservations present.	`RESDEPTH 64`

RESERVATIONDEPTH[X]	<INTEGER>	1	specifies how many priority reservations are allowed in the associated reservation stack	`RESERVATIONDEPTH[0] 4`
				`RESERVATIONQOSLIST[0] 1 3 5`
				
				(jobs with QOS values of 1, 3, or 5 can have a cumulative total of up to 4 priority reservations)

RESERVATIONPOLICY	one of the following:
	CURRENTHIGHEST, **HIGHEST**, **NEVER**
		CURRENTHIGHEST	specifies how Maui reservations will be handled. (See also RESERVATIONDEPTH)	`RESERVATIONPOLICY CURRENTHIGHEST`
				`RESERVATIONDEPTH 2`
				
				(Maui will maintain reservations for only the two currently highest priority jobs)

RESERVATIONQOSLIST[X]	one or more QOS values or [ALL]	[ALL]	specifies which QOS levels have access to the associated reservation stack	`RESERVATIONDEPTH[0] 4`
				`RESERVATIONQOSLIST[0] 1 3 5`
				
				(jobs with QOS values of 1, 3, or 5 can have a cumulative total of up to 4 priority reservations)

RESERVATIONRETRYTIME[X]	[[[DD:]HH:]MM:]SS	0	Period of time Maui will continue to attempt to start a job in a reservation when job start failures are detected due to resource manager corruption	
RESOURCECAP[X]	<DOUBLE>	0 (NO CAP)	specifies the maximum allowed pre-weighted job resource priority factor	`RESOURCECAP[0] 1000` (The total resource priority factor component of a job's priority will not be allowed to exceed 1000, i.e., 'Priority = RESWEIGHT * MIN(RESOURCECAP,<RESOURCEFACTOR>) + ...)

RESOURCEAVAILABILITYPOLICY	<POLICY>[:<RESOURCETPYE>] ...
	
	where
	POLICY is one of **COMBINED**, **DEDICATED**, or **UTILIZED**
	and
	RESOURCETYPE is one of
	PROC, **MEM**, **SWAP**, or **DISK**
		xxxxxCOMBINED	specifies how Maui will evaluate resource availability on a per resource basis	`RESOURCEAVAILABILITYPOLICY`
				`DEDICATED:PROCS COMBINED:MEM`
				
				(Maui will ignore resource utilization information in locating available processors for jobs but will use both dedicated and utilized memory information in determining memory availability)

RESOURCELIMITPOLICY	<POLICY>:<ACTION>:<RESOURCE> [:<RESOURCE>]... where **POLICY** is one of **ALWAYS**, **EXTENDEDVIOLATION**, or **BLOCKEDWORKLOADONLY** where **ACTION** is one of **CANCEL**, **REQUEUE**, or **SUSPEND** where **RESOURCE** is one or more of **PROC**, **DISK**, **SWAP**, or **MEM**	no limit enforcement	specifies how the scheduler should handle job which utilize more resources than they request. **NOTE**: Only available in Maui 3.2 and higher.	`RESOURCELIMITPOLICY ALWAYS:CANCEL:MEM` (Maui will cancel all jobs which exceed their requested memory limits.)

RESWEIGHT[X]	<INTEGER>	0	all resource priority components are multiplied by this value before being added to the total job priority.	`RESWEIGHT[0] 5`
				`MEMORYWEIGHT[0] 10`
				`PROCWEIGHT[0] 100`
				`SWAPWEIGHT[0] 0`
				`RESOURCECAP[0] 2000`
				
				(the job priority resource factor will be calculated as MIN(2000,5 * (10 * JobMemory + 100 * JobProc)))

RMAUTHTYPE[X]	one of **CHECKSUM**, **PKI**, or **SECUREPORT**	**CHECKSUM**	specifies the security protocol to be used in scheduler-resource manager communication.	`RMAUTHTYPE[0] CHECKSUM` (The scheduler will require a secure checksum associated with each resource manager message)
RMNAME[X]	<STRING>	<X>	specifies name of resource manager <X>	`RMNAME[0] DevCluster` (resource manager '0' will be referred to as 'DevCluster' in maui command output and maui logs)
RMNMPORT[X]	<INTEGER>	(any valid port number)	specifies a non-default RM node manager through which extended node attribute information may be obtained	`RMNMPORT[0] 13001` (Maui will contact the node manager located on each compute node at port 13001)
RMPOLLINTERVAL	[[[DD:]HH:]MM:]SS	00:01:00	specifies interval between RM polls	`RMPOLLINTERVAL 60` (Maui will refresh its resource manager information every 60 seconds. NOTE: this parameter specifies the global poll interval for all resource managers)

RMPORT[X]	<INTEGER>	0	specifies the port on which Maui should contact the associated resource manager. The value '0' specifies to use the appropriate default port for the resource manager type selected.	`RMTYPE[0] PBS`
				`RMHOST[0] cws`
				`RMPORT[0] 20001`
				
				(Maui will attempt to contact the PBS server daemon on host cws, port 20001)

RMSERVER[X]	<HOSTNAME>	[NONE]	specifies the host on which Maui should contact the associated resource manager. An empty value specifies to use the default hostname for the resource manager selected. NOTE: this parameter is renamed **RMHOST** in Maui 3.0.6 and higher.	`RMTYPE[0] LL2`
				`RMHOST[0]`
				`RMPORT[0] 0`
				
				(Maui will attempt to contact the Loadleveler version 2 Negotiator daemon on the default host and port, as specified in the LL config files)

RMTIMEOUT[X]	<INTEGER>	15	seconds maui will wait for a response from the associated resource manager.	`RMTIMEOUT[1] 30` (Maui will wait 30 seconds to receive a response from resource manager '1' before timing out and giving up. Maui will try again on the next iteration.)

RMTYPE[X]	<RMTYPE>[:<RMSUBTYPE>] where <RMTYPE is one of the following: **LL**, **PBS**, or **WIKI** and <RMSUBTYPE> is one of **RMS**	LL	specifies type of resource manager to be contacted by Maui. NOTE: for **RMTYPE WIKI**, **RMAUTHTYPE** must be set to **CHECKSUM**.	`RMTYPE[0] PBS`
				`RMHOST[0] cluster1`
				`RMPORT[0] 15003`
				`RMTYPE[1] PBS`
				`RMHOST[1] cluster2`
				`RMPORT[1] 15004`
				(Maui will interface to two different PBS resource managers, one located on server cluster1 at port 15003 and one located on server cluster2 at port 15004)

SERVERHOST	<HOSTNAME>	[NONE]	hostname of machine on which maui will run. NOTE: this parameter MUST be specified.	`SERVERHOST geronimo.scc.edu` (Maui will execute on the host `geronimo.scc.edu`)
SERVERMODE	one of the following: **NORMAL**, **TEST**, or **SIMULATION**	NORMAL	specifies how Maui interacts with the outside world. See <Testing> for more information	`SERVERMODE SIMULATION`
SERVERNAME	<STRING>	<SERVERHOST>	specifies the name the scheduler will use to refer to itself in communication with peer daemons	`SERVERNAME mauiA`
SERVERPORT	<INTEGER> (range: 1-64000)	40559	port on which maui will open its user interface socket	`SERVERPORT 30003` (Maui will listen for client socket connections on port 30003)
SIMAUTOSHUTDOWN	<BOOLEAN>	**TRUE**	if TRUE, the scheduler will end simulations when the active queue and idle queue become empty	`SIMAUTOSHUTDOWN ON` (The scheduler simulation will end as soon as there are no jobs running and no idle jobs which could run)
SIMCPUSCALINGPERCENT	<INTEGER>	100 (no scaling)	specifies whether to increase or decrease the runtime and wallclock limit of each job in the workload trace file.	
SIMDEFAULTJOBFLAGS	zero or more of the following: **ADVRES**, **HOSTLIST**, **RESTARTABLE**, **PREEMPTEE**, **DEDICATED**, **PREEMPTOR**	[NONE]	cause Maui to force the specified job flags on all jobs supplied in the workload trace file	`SIMDEFAULTJOBFLAGS DEDICATED` (Maui will set the 'DEDICATED' job flag on all jobs loaded from the workload trace file)
SIMEXITITERATION	<INTEGER>	0 (no exit iteration)	iteration on which a Maui simulation will create a simulation summary and exit.	`SIMEXITITERATION 36000`
SIMFLAGS	zero or more of the following: **IGNHOSTLIST**, **IGNCLASS**, **IGNQOS**, **IGNMODE**, **IGNFEATURES**	[NONE]	controls how Maui handles trace based information	`SIMFLAGS IGNHOSTLIST` (Maui will ignore hostlist information specified in the workload trace file)
SIMIGNOREJOBFLAGS	zero or more of the following: **ADVRES**, **HOSTLIST**, **RESTARTABLE**, **PREEMPTEE**, **DEDICATED**, **PREEMPTOR**	[NONE]	cause Maui to ignore specified job flags if supplied in the workload trace file	`SIMIGNOREJOBFLAGS DEDICATED` (Maui will ignore the 'DEDICATED' job flag if specified in any job trace)

SIMINITIALQUEUEDEPTH	<INTEGER>	16	specifies how many jobs the simulator will initially place in the idle job queue	`SIMINITIALQUEUEDEPTH 64`
				`SIMJOBSUBMISSIONPOLICY CONSTANTJOBDEPTH`
				(Maui will initially place 64 idle jobs in the queue and, because of the specified queue policy, will attempt to maintain this many jobs in the idle queue throughout the duration of the simulation)

SIMJOBSUBMISSIONPOLICY	one of the following: **NORMAL**, **CONSTANTJOBDEPTH**, or **CONSTANTPSDEPTH**	**CONSTANTJOBDEPTH**	specifies how the simulator will submit new jobs into the idle queue. (NORMAL mode causes jobs to be submitted at the time recorded in the workload trace file, CONSTANTJOBDEPTH and CONSTANTPSDEPTH attempt to maintain an idle queue of <SIMINITIALQUEUEDEPTH> jobs and procseconds respectively)	`SIMJOBSUBMISSIONPOLICY NORMAL` (Maui will submit jobs with the relative time distribution specified in the workload trace file.)
SIMNODECONFIGURATION	one of the following: **UNIFORM** or **NORMAL**	NORMAL	specifies whether or not maui will filter nodes based on resource configuration while running a simulation	
SIMNODECOUNT	<INTEGER>	0 (no limit)	specifies the maximum number of nodes maui will load from the simulation resource file	
SIMRESOURCETRACEFILE	<STRING>	traces/resource.trace	specifies the file from which maui will obtain node information when running in simulation mode. Maui will attempt to locate the file relative to <MAUIHOMEDIR> unless specified as an absolute path	`SIMRESOURCETRACEFILE traces/nodes.1` (Maui will obtain node traces when running in simulation mode from the <MAUIHOMEDIR>`/traces/nodes.1` file)
SIMRMRANDOMDELAY	<INTEGER>	0	specifies the random delay added to the RM command base delay accumulated when making any resource manager call in simulation mode	`SIMRMRANDOMDELAY 5` (Maui will add a random delay of between 0 and 5 seconds to the simulated time delay of all RM calls)
SIMSTOPITERATION	<INTEGER>	0 (no stop iteration)	specifies on which scheduling iteration a maui simulation will stop and was for a command to resume scheduling	`SIMSTOPITERATION 1` (Maui should stop after the first iteration of simulated scheduling and wait for admin commands)
SIMTIMERATIO	<INTEGER>	0 (no time ratio)	determines wall time speedup. Simulated Maui time will advance <SIMTIMERATIO> * faster than real wall time.	`SIMTIMERATIO 10` (Maui simulation time will advance 10 times faster than real world wall time. For example, in 1 hour, Maui will process 10 hours of simulated workload)

SIMWORKLOADTRACEFILE	<STRING>	traces/workload.trace	specifies the file from which maui will obtain job information when running in simulation mode. Maui will attempt to locate the file relative to <MAUIHOMEDIR> unless specified as an absolute path	`SIMWORKLOADTRACEFILE traces/jobs.2` (Maui will obtain job traces when running in simulation mode from the <MAUIHOMEDIR>/traces/jobs.2 file)
SRACCESS[X]	DEDICATED or SHARED	DEDICATED	If set to SHARED, allows a standing reservation to utilize resources already allocated to other non-job reservations. Otherwise, these other reservations will block resource access. (See Managing Reservations)	`SRACCESS[2] SHARED` (Standing reservation '2' may access resources allocated to existing standing and administrative reservations)
SRACCOUNTLIST[X]	list of valid account names	[NONE]	specifies that jobs with the associated accounts may use the resources contained within this reservation	`SRACCOUNTLIST[1] ops staff` (jobs using the account `ops` or `staff` are granted access to the resources in standing reservation '1')
SRCHARGEACCOUNT[X]	any valid accountname	[NONE]	specifies the account to which maui will charge all idle cycles within the reservation (via the allocation bank)	`SRCHARGEACCOUNT[1] steve` (Maui will charge all idle cycles within reservations supporting standing reservation 1 to user 'steve')

SRCFG[X]	one or more of the following <ATTR>=<VALUE> pairs
	ACCOUNTLIST
	CHARGEACCOUNT
	CLASSLIST
	DAYS
	DEPTH
	ENDTIME
	FLAGS
	GROUPLIST
	HOSTLIST
	JOBATTRLIST
	NODEFEATURES
	PARTITION
	PERIOD
	PRIORITY
	PROCLIMIT
	QOSLIST
	RESOURCES
	STARTTIME
	TASKCOUNT
	TASKLIMIT
	TIMELIMIT
	TPN
	USERLIST
	WENDTIME
	WSTARTTIME
		[NONE]	specifies attributes of a standing reservation. Available in Maui 3.2 and higher. See Managing Reservations for details.	`SRCFG[fast] STARTTIME=9:00:00 ENDTIME=15:00:00`
				`SRCFG[fast] HOSTLIST=node0[1-4]$`
				`SRCFG[fast] QOSLIST=high:low`
				(Maui will create a standing reservation running from 9:00 AM to 3:00 PM on nodes 1 through 4 accessible by jobs with QOS high or low.)

SRCLASSLIST[X]	list of valid class names	[NONE]	specifies that jobs requiring any of these classes may use the resources contained within this reservation	`SRCLASSLIST[2] interactive` (maui will allow all jobs requiring any of the classes listed access to the resources reserved by standing reservation '2')
SRDAYS[X]	one or more of the following (space delimited) Mon Tue Wed Thu Fri Sat Sun or [ALL]	[ALL]	specifies which days of the week the standing reservation will be active	`SRDAYS[1] Mon Tue Wed Thu Fri` (standing reservation '1' will be active on Monday thru Friday)
SRDEPTH[X]	<INTEGER>	2	specifies the number of standing reservations which will be created (one per day)	`SRDEPTH[1] 7` (specifies that standing reservations will be created for standing reservation '1' for today, and the next 6 days)

SRENDTIME[X]	[[HH:]MM:]SS	24:00:00	specifies the time of day the standing reservation becomes inactive	`STSTARTTIME[2] 8:00:00`
				`SRENDTIME[2] 17:00:00`
				(standing reservation '2' is active from 8:00 AM until 5:00 PM)

SRFEATURES[X]	space delimited list of node features	[NONE]	specifies the required node features for nodes which will be part of the standing reservation	`SRFEATURES[3] wide fddi` (all nodes used in the standing reservation must have both the 'wide' and 'fddi' node attributes)
SRFLAGS	colon delimited list of zero or more of the following flags: **SINGLEUSE*** **BYNAME** **PREEMPTEE*** **SLIDEFORWARD*** **FORCE** (only enabled in Maui 3.2 and later)	[NONE]	specifes special reservation attributes. See Managing Reservations for details.	`SRFLAGS[1] BYNAME` (Jobs may only access the resources within this reservation if they explicitly request the reservation 'by name'

SRGROUPLIST[X]	one or more space delimited group names	[ALL]	specifies the groups which will be allowed access to this standing reservation	`SRGROUPLIST[1] staff ops special`
				`SRCLASSLIST[1] interactive`
				(Maui will allow jobs with the listed group ID's or which request the job class 'interactive' to use the resources covered by standing reservation 1.)

SRHOSTLIST[X]	one or more space delimited host names	[ALL]	specifies the set of host from which Maui can search for resources to satisfy the reservation. If SRTASKCOUNT is also specified, only <SRTASKCOUNT> tasks will be reserved. Otherwise, all hosts listed will be reserved.	`SRHOSTLIST[3] node001 node002 node003`
				`SRRESOURCES[3] PROCS=2;MEM=512`
				`SRTASKCOUNT[3] 2`
				(Maui will reserve 2 tasks - with 2 processors and 512 MB each, using resources located on node001, node002, and/or node003)

SRMAXTIME[X]	[[[DD:]HH:]MM:]SS	-1 (no time based access)	specifies the maximum allowed overlap between a the standing reservation and a job requesting resource access	`SRMAXTIME[6] 1:00:00` (Maui will allow jobs to access up to one hour of resources in standing reservation 6)
SRNAME[X]	<STRING>	[NONE]	specifies name of standing reservation <X>	`SRNAME[1] interactive` (The name of standing reservation '1' is 'interactive')

SRPARTITION[X]	<STRING>	[ALL]	specifies the partition in which the standing reservation should be created	`SRPARTITION[0] OLD` (only select resource for standing reservation 0 in partition 'OLD')
SRPERIOD[X]	one of **DAY**, **WEEK**, or **INFINITY**	**DAY**	specifies the periodicity of the standing reservation	`SRPERIOD[1]  WEEK` (each standing reservation covers a one week period)
SRQOSLIST[X]	zero or more valid QOS names	[NONE]	specifies that jobs with the listed QOS names can access the reserved resources	`SRQOSLIST[1] 1 3 4 5` (maui will allow jobs using QOS 1, 3, 4, and 5 to use the reserved resources)
SRRESOURCES[X]	semicolon delimited <ATTR>=<VALUE> pairs	PROCS=-1 (All processors available on node)	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 node) Supported resources currently include the following: PROCS (number of processors) MEM (real memory in MB) DISK (local disk in MB) SWAP (virtual memory in MB)	`SRRESOURCES[1] PROCS=1;MEM=512` (each standing reservation task will reserve one processor and 512 MB of real memory)

SRSTARTTIME[X]	[[HH:]MM:]SS	00:00:00	specifies the time of day the standing reservation becomes active	`SRSTARTTIME[1] 08:00:00`
				`SRENDTIME[1]  17:00:00`
				(standing reservation '1' is active from 8:00 AM until 5:00 PM)

SRTASKCOUNT[X]	<INTEGER>	0	specifies how may tasks should be reserved for the reservation	`SRRESOURCES[2] PROCS=1;MEM=256`
				`SRTASKCOUNT[2] 16`
				(standing reservation '2' will reserve 16 tasks worth of resources, in this case, 16 procs and 4 GB of real memory)

SRTIMELOGIC[X]	**AND** or **OR**	**OR**	specifies how SRMAXTIME access status will be combined with other standing reservation access methods to determine job access. If SRTIMELOGIC is set to OR, a job is granted access to the reserved resources if it meets the MAXTIME criteria or any other access criteria (i.e., SRUSERLIST) If SRTIMELOGIC is set to AND, a job is granted access to the reserved resources only if it meets the MAXTIME criteria and at least on other access criteria	`SRMAXTIME[5]  1:00:00`
				`SRUSERLIST[5]  carol charles`
				`SRTIMELOGIC[5] AND`
				(Maui will allow jobs from users carol and charles to use up to one hour of resources in standing reservation 5)

SRTPN[X]	<INTEGER>	0 (no TPN constraint)	specifies the minimum number of tasks per node which must be available on eligible nodes.	`SRTPN[2]  4`
				`SRRESOURCES[2] PROCS=2;MEM=256`
				(Maui must locate at least 4 tasks on each node that is to be part of the reservation. That is, each node included in standing reservation '2' must have at least 8 processors and 1 GB of memory available)

SRUSERLIST[X]	space delimited list of users	[NONE]	specifies which users have access to the resources reserved by this reservation	`SRUSERLIST[1] bob joe mary` (users bob, joe and mary can all access the resources reserved within this reservation)

SRWENDTIME[X]	[[[DD:]HH:]MM:]SS	7:00:00:00	specifies the week offset at which the stand reservation should end	`SRSTARTTIME[1] 1:08:00:00`
				`SRENDTIME[1]  5:17:00:00`
				(standing reservation '1' will run from Monday 8:00 AM to Friday 5:00 PM)

SRWSTARTTIME[X]	[[[DD:]HH:]MM:]SS	0:00:00:00	specifies the week offset at which the standing reservation should start	`SRSTARTTIME[1] 1:08:00:00`
				`SRENDTIME[1]  5:17:00:00`
				(standing reservation '1' will run from Monday 8:00 AM to Friday 5:00 PM)

STATDIR	<STRING>	stats	specifies the directory in which Maui statistics will be maintained	`STATDIR  /var/adm/maui/stats`
SYSCFG	list of zero or more space delimited <ATTR>=<VALUE> pairs where <ATTR> is one of the following: **PRIORITY**, **FSTARGET**, **QLIST**, **QDEF**, **PLIST**, **PDEF**, **FLAGS**, or a fairness policy specification.	[NONE]	specifies system-wide default attributes. See the Attribute/Flag Overview for more information. **NOTE:** Only available in Maui 3.0.7 and higher.	`SYSCFG PLIST=Partition1 QDEF=highprio` (by default, all jobs will have access to partition `Partition1` and will use the QOS `highprio`)
SWAPWEIGHT	<INTEGER>	0	specifies the priority weight assigned to the virtual memory request of a job	`SWAPWEIGHT  10`
SYSTEMDEFAULTJOBWALLTIME	[[[DD:]HH:]MM:]SS	10:00:00:00	specifies the walltime for jobs which do not explicitly set this value	`SYSTEMDEFAULTJOBWALLTIME 1:00:00:00` (Maui will assign a wallclock limit of 1 day to jobs which do not explicitly specify a wallclock limit)
SYSTEMMAXPROCPERJOB	<INTEGER>	-1 (NO LIMIT)	specifies the maximum number of processors that can be requested by any single job	`SYSTEMMAXJOBPROC 256` (Maui will reject jobs requesting more than 256 processors)
SYSTEMMAXPROCSECONDPERJOB	<INTEGER>	-1 (NO LIMIT)	specifies the maximum number of proc-seconds that can be requested by any single job	`SYSTEMMAXJOBPROCSECOND 86400` (Maui 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	[[[DD:]HH:]MM:]SS	-1 (NO LIMIT)	specifies the maximum amount of wallclock time that can be requested by any single job	`SYSTEMMAXJOBWALLTIME 1:00:00:00` (Maui will reject jobs requesting more than one day of walltime)
TARGWEIGHT	<INTEGER>	0	specifies the weight to be applied to a job's queuetime and expansion factor target components	`TARGETWEIGHT  1000`

TASKDISTRIBUTIONPOLICY	one of **DEFAULT** or **LOCAL**	**DEFAULT**	specifies how job tasks should be mapped to allocated resources.	`TASKDISTRIBUTIONPOLICY DEFAULT` (Maui should use standard task distribution algorithms)
TRAPFUNCTION	<STRING>	[NONE]	specifies the functions to be trapped	`TRAPFUNCTION UpdateNodeUtilization|GetNodeSResTime`
TRAPJOB	<STRING>	[NONE]	specifies the jobs to be trapped	`TRAPJOB buffy.0023.0`
TRAPNODE	<STRING>	[NONE]	specifies the nodes to be trapped	`TRAPNODE node001|node004|node005`
TRAPRES	<STRING>	[NONE]	specifies the reservations to be trapped	`TRAPRES interactive.0.1`
USAGEWEIGHT	<INTEGER>	0	specifies the weight assigned to the percent and total job usage subfactors	`USAGEWEIGHT 100`
USAGEPERCENTWEIGHT	<INTEGER>			
USEMACHINESPEED	ON or OFF	OFF	specifies whether or not job wallclock limits should be scaled by the machine speed of the node(s) they are running on.	`USEMACHINESPEED ON` (job <X> specifying a wallclock limit of 1:00:00 would be given only 40 minutes to run if started on a node with a machine speed of 1.5)
USERCFG[<USERID>]	list of zero or more space delimited <ATTR>=<VALUE> pairs where <ATTR> is one of the following: **PRIORITY**, **FSTARGET**, **QLIST**, **QDEF**, **PLIST**, **PDEF**, **FLAGS**, or a fairness policy specification.	[NONE]	specifies user specific attributes. See the flag overview for a description of legal flag values. **NOTE:** Only available in Maui 3.0.7 and higher.	`USERCFG[john] MAXJOB=50 QDEF=highprio` (up to 50 jobs submitted under the user ID `john` will be allowed to execute simultaneously and will be assigned the QOS `highprio` by default.)
USERWEIGHT	<INTEGER>	0	specifies the weight assigned to the specified user priority (see Credential Priority Factor)	`USERWEIGHT 100`
USESYSTEMQUEUETIME	ON or OFF	OFF	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 (ON) or the time the job has been idle (OFF). NOTE: In Maui 3.0.8 and higher, this parameter has been superseded by the JOBPRIOACCRUALPOLICY parameter.	`USESYSTEMQUEUETIME OFF` (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.) (See QUEUETIMEFACTOR for more info)
XFCAP	<DOUBLE>	0 (NO CAP)	specifies the maximum total pre-weighted contribution to job priority which can be contributed by the expansion factor component. This value is specified as an absolute priority value, not as a percent.	`XFCAP 10000` (Maui will not allow a job's pre-weighted XFactor priority component to exceed the value 10000)
XFMINWCLIMIT	[[[DD:]HH:]MM:]SS	-1 (NO LIMIT)	specifies the minimum job wallclock limit that will be considered in job expansion factor priority calculations	`XFMINWCLIMIT 0:01:00` (jobs requesting less than one minute of wallclock time will be treated as if their wallclock limit was set to one minute when determining expansion factor for priority calculations)

XFWEIGHT	<INTEGER>	0	specifies the weight to be applied to a job's minimum expansion factor before it is added to the job's cumulative priority	`XFWEIGHT 1000` (Maui will multiply a job's XFactor value by 1000 and then add this value to its total priority

Appendix G: Commands Overview

Command	Description
canceljob	cancel job
changeparam	change in memory parameter setting
checkjob	provide detailed status report for specified job
checknode	provide detailed status report for specified node
diagnose	provide diagnostic report for various aspects of resources, workload, and scheduling
mjobctl	control and modify job
mnodectl	control and modify nodes
mprof	profile historical system performance
releasehold	release job defers and holds
releaseres	release reservations
resetstats	reset scheduler statistics
runjob	force a job to run immediately
schedctl	manage scheduler activity
sethold	set job holds
setqos	modify job QOS settings
setres	set an admin/user reservation
setspri	adjust system priority of jobs
showbf	show backfill window - show resources available for immediate use
showconfig	show current scheduler configuration
showgrid	show various tables of scheduling/system performance
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

G.1: canceljob

---

canceljob *JOB* [*JOB*] ... [-h]

Purpose

Cancels the specified job(s).

Permissions

This command can be run by any Maui Scheduler Administrator and the owner of the job.

Parameters

JOB Job name you want to cancel.

Flags

-h Show help for this command.

Description

The `canceljob` command is used to selectively cancel the specified job(s) (active, idle, or non-queued) from the queue.

Example 1

```
% canceljob -h
```

Shows help for this command.

Example 2

```
% canceljob fr1n04.981.0
```

Cancels job 981 running on Frame 1, Node 04.

Related Commands

This command is equivalent to the LoadLeveler `llcancel` command.

You can find job numbers with the `showq` command.

Default File Location

```
/u/loadl/maui/bin/canceljob
```

Notes

None.

changeparam

Overview:

The **changeparam** command is used to dynamically change the value of any parameter which can be specified in the `maui.cfg` file. The changes take affect at the beginning of the next scheduling iteration. They are not persistent, only lasting until Maui is shutdown.

Format:

changeparam <PARAMETER> <VALUE>

<PARAMETER> is any valid Maui parameter
<VALUE> is any valid value for <PARAMETER>

Flags:

[NONE]

Access:

This command can be run by any user with 'ADMIN1' authority.

Example:

Set Maui's LOGLEVEL to 6 for the current run:

> changeparam LOGLEVEL 6

parameters changed

Example:

Set Maui's ADMIN1 userlist to 'sys mike peter'

> changeparam ADMIN1 sys mike peter

parameters changed

checkjob

---

checkjob [ARGS] <*JOBID*>

Purpose

Display detailed job state information and diagnostic output for specified job.

Permissions

This command can be run by any Maui admininstrator. Additionally, valid users may use this command to obtain information about their own jobs.

Args	Details
-A	provide output in the form of parsable *Attribute*-Value pairs
-h	display command usage *help*
-l <POLICYLEVEL>	check job start eligibility subject to specified throttling *policy* level. <POLICYLEVEL> can be one of **HARD**, **SOFT**, or **OFF**
-r <RESID>	check job access to specified *reservation*
-v	display *verbose* job state and eligibility information

Description

This command allows any Maui administrator to check the detailed status and resources requirements of a job. Additionally, this command performs numerous diagnostic checks and determines if and where the could potentially run. Diagnostic checks include policy violations (See the Throttling Policy Overview for details), reservation constraints, 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.

If a job cannot run, 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	
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' which 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:

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

The checkjob command displays the following job attributes:

Attribute	**Value**	**Description**
Account	<STRING>	Name of account associated with job
Actual Run Time	[[[DD:]HH:]MM:]SS	Length of time job actually ran. **NOTE**: This info only display in simulation mode.
Arch	<STRING>	Node architecture required by job
Class	[<CLASS NAME> <CLASS COUNT>]	Name of class/queue required by job and number of class initiators required per task.
Dedicated Resources Per Task	<XXX>	
Disk	<INTEGER>	Amount of local disk required by job (in MB)
Exec Size	<INTEGER>	Size of job executable (in MB)
Executable	<STRING>	Name of job executable
Features	Square bracket delimited list of <STRING>s	Node features required by job
Group	<STRING>	Name of UNIX group associated with job

Holds	Zero of more of User, System, and Batch	Types of job holds currently applied to job
Image Size	<INTEGER>	Size of job data (in MB)
Memory	<INTEGER>	Amount of real memory required per node (in MB)
Network	<STRING>	Type of network adapter required by 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
QOS	<STRING>	Quality of Service associated with job
QueueTime	<TIME>	Time job was submitted to resource management system
StartCount	<INTEGER>	Number of times job has been started by Maui
StartPriority	<INTEGER>	Start priority of job
State	One of Idle, Starting, Running, etc	Current Job State
Total Tasks	<INTEGER>	Number of tasks requested by job
User	<STRING>	Name of user submitting job
WallTime:	[[[DD:]HH:]MM:]SS	Length of time job has been running
WallTime Limit:	[[[DD:]HH:]MM:]SS	Maximum walltime allowed to job

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

Examples

Example 1

```
> checkjob -v job05

checking job job05

State: Idle  (User: john  Group: staff  Account: [NONE])
WallTime: 0:00:00  (Limit: 6:00:00)

Submission Time: Mon Mar  2 06:34:04

Total Tasks: 2
```

```
Req[0]  TaskCount: 2  Partition: ALL
Network: hps_user  Memory >= 0  Disk >= 0  Features: [NONE]
Opsys: AIX43  Arch: R6000  Class: [batch 1]
ExecSize: 0  ImageSize: 0
Dedicated Resources Per Task: Procs: 1
NodeCount: 0

IWD: [NONE]     Executable:  cmd
QOS: [DEFAULT]  Bypass: 0  StartCount: 0
Partition Mask: ALL
Holds:    Batch
batch hold reason:  Admin
PE:  2.00  StartPriority:  1
job cannot run  (job has hold in place)
job cannot run  (insufficient idle procs:  0 available)
----
```

Note that the example job cannot be started for two different reasons.

- It has a batch hold in place.
- There are no idle resources currently available

See also:

diagnose -j - display additional detailed information regarding jobs

---

checknode

---

checknode *NODE* [-h]

Purpose

Displays state information and statistics for the specified node.

Permissions

This command can be run by any Maui Scheduler Administrator.

Parameters

NODE Node name you want to check.

Flags

-h Help for this command.

Description

This command shows detailed state information and statistics for nodes that run jobs (those running `LoadL_startd`).

NOTE

This command returns an error message if it is run against a scheduling node (one running `schedd`).

The following information is returned by this command:

Disk	Disk space available
Memory	Memory available
Swap	Swap space available
State	Node state
Opsys	Operating system
Arch	Architecture
Adapters	Network adapters available
Features	Features available
Classes	Classes available
Frame	IBM SP frame number associated with node
Node	IBM SP node number associated with node
StateTime	Time node has been in current state in HH:MM:SS notation
Downtime	Displayed only if downtime is scheduled
Load	CPU Load (Berkley one-minute load average)
TotalTime	Total time node has been detected since statistics initialization expressed in HH:MM:SS notation
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)

BusyTime	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)

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

Example

```
% checknode fr26n10

Checking Node fr26n10.mhpcc.edu

Disk (KB):     2076  Memory (MB):      512  Swap (KB): 470772
State:         Down  Opsys:          AIX41  Arch:       R6000
Adapters: [ethernet]
Features: [Thin][Dedicated]
Classes:  [batch][medium]
Frame:    26  Node: 10

StateTime:  Node has been in current state for  5:02:23
DownTime:  (-26844 Seconds/-7.46 Hours) Thu Sep  4 09:00:00
Load:     0.009
TotalTime:   30:18:29  UpTime:   23:28:51 (77.47%) BusyTime:   19:21:46 (63.89%)
```

Related Commands

Further information about node status can be found using the `showstate` command.

You can determine scheduling nodes with the LoadLeveler `llstatus` command (nodes that have `Avail` in the `Schedd` column).

Default File Location

`/u/loadl/maui/bin/checknode`

Notes

None.

---

diagnose (Under Construction)

Overview:

The 'diagnose' command is used to display information about various aspects of scheduling and the results of internal diagnostic tests

Format:

```
    diagnose [ -a [<ACCOUNTID>]]          // Diagnose Accounts
             [ -f ]                       // Diagnose Fairshare
             [ -g [<GROUPID>]]            // Diagnose Groups
             [ -j [<JOBID>]]              // Diagnose Job
             [ -m ]                       // Diagnose Frames
             [ -n [ -t <PARTITION>] [<NODEID>]]  // Diagnose
Nodes
             [ -p [ -t <PARTITION>]]      // Diagnose Priority
             [ -q [ -l <POLICYLEVEL>]]    // Diagnose Job Queue
             [ -Q ]                       // Diagnose QOS
Configuration
             [ -r ]                       // Diagnose
Reservations
             [ -t ]                       // Diagnose
Partitions
             [ -u [<USERID>]]             // Diagnose Users
```

Flags:

-a Show detailed information about accounts

-f Show detailed information about fairshare configuration and status

-j Show detailed information about jobs

Example:

> diagnose -r

---

profiler

XXX INFO NOT YET AVAILABLE

Purpose

XXX

Permissions

This command can be run by any Maui Scheduler Administrator.

Parameters

Flags

Description

Example

Related Commands

Default File Location

`/u/loadl/bqs/bin/`

Notes

None.

releasehold

---

releasehold [-h|-a|-b] ***JOBEXP***

Purpose

Release hold on specified job(s).

Permissions

This command can be run by any Maui Scheduler Administrator.

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.

Description

This command allows you to release batch holds or all holds (system, user, and batch) on specified jobs. Any number of jobs may be released with this command.

Example 1

```
> releasehold -b fr17n02.1072.0
Batch hold released on all specified jobs
```

In this example, a batch hold was released from this one job.

Example 2

```
> releasehold -a fr17n02.1072.0 fr15n03.1017.0
All holds released on all specified jobs
```

In this example, all holds were released from these two jobs.

Related Commands

You can place a hold on a job using the `sethold` command.

Notes

None.

releaseres

---

releaseres [ARGUMENTS] <RESERVATION ID> [<RESERVATION ID]...

ARGUMENTS:

[-h] // USAGE HELP

Purpose

Release existing reservation.

Access

Users can use this command to release any reservation they own. Level 1 and level 2 Maui administrators may use this command to release any reservation.
This command can be run by any user.

Parameters

RESERVATION ID	Name of reservation to release.

Flags

-h Help for this command.

Description

This command allows Maui Scheduler Administrators 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.

Example

Release two existing reservations.

```
% releaseres system.1 bob.2
released User reservation 'system.1'
released User reservation 'bob.2'
```

Related Commands

You can view reservations with the `showres` command.

You can set a reservation using the `setres` command.

Notes

See the Reservation document for more information.

---

resetstats

resetstats [-h]

Purpose

Resets statistics to start-up state.

Permissions

This command can be run by any Maui Scheduler Administrator.

Parameters

None.

Flags

-h Help for this command.

Description

This command resets all internally-stored Maui Scheduler statistics to the initial start-up state as of the time the command was executed.

Example

```
% resetstats

Statistics Reset at time Wed Feb 25 23:24:55 1998
```

Related Commands

None.

Default File Location

`/u/loadl/maui/bin/resetstats`

Notes

None.

runjob

---

runjob [ARGS] <JOBID>

Purpose

Immediately run the specified job.

Permissions

This command can be run by any Maui administrator.

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
-h	*Help* for this command.
-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 constaints, and reservations

Description

This command will attempt to immediately start a job.

Example

```
> runjob cluster.231

job cluster.231 successfully started
```

This example attempts to run job cluster.231.

See Also:

- `canceljob` - cancel a job.
- `checkjob` - show detailed status of a job.
- `showq` - list queued jobs.

---

schedctl

Overview:

The 'schedctl' command controls various aspects of scheduling behavior. It is used to manage scheduling activity, kill the scheduler, and create resource trace files.

Format:

schedctl { -k | -n | -r [<RESUMETIME>] | { -s | -S } [<ITERATION>] }

Flags:

-k

shutdown the scheduler at the completion of the current scheduling iteration

-n

dump a node table trace to <STDOUT> (for use in simulations)

-r [<RESUMETIME>]

resume scheduling in <RESUMETIME> seconds or immediately if not specified

-s [<ITERATION>]

suspend scheduling at iteration <ITERATION> or at the completion of the current scheduling iteration if not specified. If <ITERATION> is followed by the letter 'I', maui will not process client requests until this iteration is reached.

-S [<ITERATION>]

suspend scheduling in <ITERATION> more iterations or in one more iteration if not specified. If <ITERATION> is followed by the letter 'I', maui will not process client requests until <ITERATION> more scheduling iterations have been completed.

Example:

Shut maui down

> schedctl -k

maui shutdown

Example:

Stop maui scheduling

> schedctl -s

maui will stop scheduling immediately

Example:

Resume maui scheduling

> schedctl -r

maui will resume scheduling immediately

Example:

Stop maui scheduling in 100 more iterations. Specify that maui should not respond to client requests until that point is reached.

> schedctl -S 100I

maui will stop scheduling in 100 iterations

---

sethold

---

sethold [-b | -h] *JOB* [*JOB*] [*JOB*] ...

Purpose

Set hold on specified job(s).

Permissions

This command can be run by any Maui Scheduler Administrator.

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 administrator to manually set a batch hold.

-h Help for this command.

Description

This command allows you to place a hold upon specified jobs.

Example

```
% sethold -b fr17n02.1072.0 fr15n03.1017.0

Batch Hold Placed on All Specified Jobs
```

In this example, a batch hold was placed on job fr17n02.1072.0 and job fr15n03.1017.0.

Related Commands

Release holds with the releasehold command.

Default File Location

```
/u/loadl/maui/bin/sethold
```

Notes

None.

setqos

---

setqos [-h] ***QOS JOB***

Purpose

Set Quality Of Service for a specified job.

Permissions

This command can be run by any user.

Parameters

JOB Job number.

QOS Quality Of Service level. Range is 0 (lowest) to 8 (highest). Jobs default to a QOS level of 0, unless the user, group, or account has a different value specified in the fairshare configuration file (`fs.cfg`). Users are allowed to set the QOS for their own jobs in the range of 0 to the maximum value allowed by the user, group, and/or account which owns the job.

Flags

-h Help for this command.

Description

This command allows you to set the Quality Of Service (QOS) level for a specified job. Users are allowed to use this command to change the QOS of their own jobs.

Example

```
% setqos 3 fr28n13.1198.0

Job QOS Adjusted
```

This example sets the Quality Of Service to a value of 3 for job number fr28n13.1198.0.

Related Commands

None.

Default File Location

```
/u/loadl/maui/bin/setqos
```

Notes

None.

setres

setres [ARGUMENTS] <RESOURCE_EXPRESSION>

ARGUMENTS:

[-a <ACCOUNT_LIST>]
[-c <CHARGE_SPEC>]
[-d <DURATION>]
[-e <ENDTIME>]
[-f <FEATURE_LIST>]*
[-g <GROUP_LIST>]
[-h] // USAGE HELP
[-n <NAME>]
[-p <PARTITION>]
[-q <QUEUE_LIST>]* // (ie CLASS_LIST)
[-Q <QOSLIST>]
[-r <RESOURCE_DESCRIPTION>]
[-s <STARTTIME>]
[-u <USER_LIST>]
[-x <FLAGS>]

* NOTE: only available in Maui 3.2 and higher.

Purpose

Reserve resources for use by jobs with particular credentials or attributes.

Access

This command can be run by level 1 and level 2 Maui administrators.

Parameters

Name	Format	Default	Description
ACCOUNT_LIST	<STRING>[:<STRING>]...	[NONE]	list of accounts that will be allowed access to the reserved resources

CHARGE_SPEC	<ACCOUNT>[,<GROUP>[,<USER>]]	[NONE]	specifies which credentials will be accountable for unused resources dedicated to the reservation
CLASS_LIST	<STRING>[:<STRING>]...	[NONE]	list of classes that will be allowed access to the reserved resource
DURATION	[[[DD:]HH:]MM:]SS	[[[DD:]HH:]MM:]SS	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)
FEATURE_LIST	<STRING>[:<STRING>]...	[NONE]	list of node features which must be possessed by the reserved resources
FLAGS	<STRING>[:<STRING>]...	[NONE]	list of reservation flags (See Managing Reservations for details)

GROUP_LIST	<STRING>[:<STRING>]...	[NONE]	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
PARTITION	<STRING>	[ANY]	partition in which resources must be located
QOS_LIST	<STRING>[:<STRING>]...	[NONE]	list of QOS's that will be allowed access to the reserved resource
RESOURCE_ DESCRIPTION	colon delimited list of zer or more of the following <ATTR>=<VALUE> pairs **PROCS**=<INTEGER> **MEM**=<INTEGER> **DISK**=<INTEGER> **SWAP**=<INTEGER>	PROCS=-1	specifies the resources to be reserved per task. (-1 indicates all resources on node)
RESOURCE_ EXPRESSION	**ALL** or **TASKS**{==|>=}<TASKCOUNT> or	Required Field. No Default	specifies the tasks to reserve. ALL indicates all resources available should be reserved. **NOTE**: if ALL or a host expression is specified, Maui will apply the reservation

	<HOST_REGEX>		regardless of existing reservations and exclusitivity issues. If TASKS is used, Maui will only allocate accessible resources.
STARTTIME	[HH[:MM[:SS]]][_MO[/DD[/YY]]] or +[[[DD:]HH:]MM:]SS	[NOW]	absolute or relative time reservation will start
USER_LIST	<STRING>[:<STRING>]...	[NONE]	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 which meet the specifed 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 the Reservation Overview for more information.

Reservations can be viewed using the showres command and can be released using the releaseres command.

Example 1

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 2

Schedule a system wide reservation to allow a system maintenance on Jun 20, 8:00 AM until Jun 22, 5:00 PM.

% setres -s 8:00:00_06/20 -e 17:00:00_06/22 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

Reserve one processor and 512 MB of memory on nodes node003 through node 006 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
```

Related Commands

Use the showres command to view reservations.
Use the releaseres command to release reservations.
Use the diagnose -r command to analyze and present detailed information about reservations.

---

setspri

---

setspri *PRIORITY* [-r] *JOB*

Purpose

Set or remove and absolute or relative system priority on a specified job.

Permissions

This command can be run by any Maui Scheduler Administrator.

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. 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 +/- 1000000000 is acceptable.

Flags

-h Help for this command.

-r Set relative system priority on job.

Description

This command allows 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.

Example 1

```
% setspri 10 fr13n03.24.0

Job System Priority Adjusted
```

In this example, a system priority of 10 is set for job fr13n03.24.0.

Example 2

```
% setspri 0 fr13n03.24.0

Job System Priority Adjusted
```

In this example, system priority is cleared for job fr13n03.24.0.

Example 3

```
> setspri -r 100000 job.00001

Job System Priority Adjusted
```

In this example, the job's priority will be increased by 100000 over the value determine by configured priority policy.

Related Commands

Use the checkjob command to check the system priority level, if any, for a given job.

Notes

None.

showbf

---

showbf
[-A] // show information accessible by (A)ny user, group, or account
[-a *ACCOUNT*]
[-c *CLASS*]
[-d *DURATION*]
[-f *FEATURELIST*]
[-g *GROUP*]
[-h]
[-m ['*MEMCMP*] *MEMORY*']
[-n *NODECOUNT*]
[-p *PARTITION*]
[-q *QOS*]
[-u *USER*]
[-v] *// VERBOSE*

Purpose

Shows what resources are available for immediate use. NOTE: if specific information is not specified, showbf will return information for the user and group running but with global access for other fields. For example, if '-q <QOS>' is not specified, Maui will return backfill information for a job which could *magically* access all QOS based resources (ie, resources covered by reservations with a QOS based ACL), if '-c <CLASS>' is not specifed, it will return the info for resources accessible to any class.

Permissions

This command can be run by any user.

Parameters

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.
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.
QOS	Specify QOS to check with -q flag.
USER	Specify particular user to check with -u flag.
PARTITION	Specify partition to check with -p flag.

Flags

-A Show backfill information for all users, groups, and accounts. By default, showbf uses the default user, group, and account ID of the user issuing the showbf command.

-a Show backfill information only for specified account.

-d Show backfill information for specified duration.

-g Show backfill information only for specified group.

-h Help for this command.

-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 backfill information for a specified number of nodes. That is, this flag can be used to force `showbf` to display only windows larger than a specified size.

-p Show backfill information for the specified partition.

-q Show information for the specified QOS.

-u Show backfill information only for specified user.

Description

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 thus obtain quick job turnaround times. This command incorporates down time, reservations, and node state information in determining the available backfill window.

Example 1

```
% showbf
backFill window (user: 'john' group: 'staff' partition: ALL) Mon Feb 16 08:28:54

partition FAST:
   9 procs available for  4:54:18

partition SLOW:
  34 procs available for 10:25:30
  26 procs available for  7:00:19
   1 proc  available with no timelimit
```

In this example, a job requiring up to 34 processors could be submitted for immediate execution in partition 2 as long as it required less than 10 hours, 25 minutes. Likewise, jobs requiring up to 26 processors that complete in less than 7 hours could also run in partition SLOW. A single-processor job with arbitrary wallclock limits could also run in this partition.

In this example, the window is specifically for user `john` in group `staff`. This information is important because processors can be reserved for particular users and groups, thus causing backfill windows to be different for each person. Backfill window information for a non-default user, group, and/or account can be displayed using the `-u, -g,` and `-a` flags, respectively. A backfill window with global user, group, and account access can be displayed using the `-A` flag.

Example 2

```
% showbf -r 16 -d 3:00:00

backFill window (user: 'john' group: 'staff' partition: ALL) Mon Feb 16 08:28:54
partition ALL:
 33 procs available with no time limit
```

In this example, 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 SRMAXTIME)

Example 3

```
% showbf -m '>128'
backfill window (user: 'john' group: 'staff' partition: ALL) Thu Jun 18 16:03:04

no procs available
```

In this example, a backfill window is requested consisting for available processors located only on nodes with over 128 MB of memory. Unfortunately, in the example, no processors are available which meet this criteria at the present time. :(

Related Commands

Use the showq command to show jobs in the various queues.

Use the diagnose command to show the partitions.

Notes

See the Backfill document for more information.

---

showconfig

---

showconfig [-v] [-h]

Purpose

View the current configurable parameters of the Maui Scheduler.

Permissions

This command can be run by a level 1, 2, or 3 Maui administrator.

Parameters

None.

Flags

-h Help for this command.

-v This optional flag turns on verbose mode, which shows all possible Maui Scheduler parameters and their current settings. If this flag is not used, this command operates in context-sensitive terse mode, which shows only relevant parameter settings.

Description

The showconfig command shows the current scheduler version and the settings of all 'in memory' parameters. These parameters are set via internal defaults, command line arguments, environment variable settings, parameters in the maui.cfg file, and via the changeparam command. Because of the many sources of configuration settings, the output may differ from the contents of the maui.cfg file. The output is such that it can be saved and used as the contents of the maui.cfg file if desired.

Example

```
> showconfig
# maui scheduler version 3.0.2.0 (PID: 11080)

BACKFILLPOLICY                  FIRSTFIT
BACKFILLMETRIC                  NODES
```

```
ALLOCATIONPOLICY                MINRESOURCE
RESERVATIONPOLICY               CURRENTHIGHEST
...
```

IMPORTANT NOTE: the showconfig flag without the '-v' flag does not show the settings of all parameters. It does show all major parameters and all parameters which are in effect and have been set to non-default values. However, it hides other rarely used parameters and those which currently have no effect or are set to default values. To show the settings of all 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 Commands

Use the changeparam command to change the various Maui Scheduler parameters.

Notes

See the Parameters document for details about configurable parameters.

---

showgrid

---

showgrid *STATISTICTYPE* [-h]

Purpose

Shows table of various scheduler statistics.

Permissions

This command can be run by any Maui Scheduler Administrator.

Parameters

STATISTICTYPE Values for this parameter:

AVGBYPASS	Average bypass count. Includes summary of job-weighted expansion bypass and total samples.
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, node-weighted expansion factor, node-second-weighted expansion factor, and total number of samples.
BFCOUNT	Number of jobs backfilled. Includes summary of job-weighted backfill job percent and total samples.
BFNHRUN	Number of node-hours backfilled. Includes summary of job-weighted backfill node-second percentage and total samples.
JOBCOUNT	Number of jobs. Includes summary of total jobs and total samples.
JOBEFFICIENCY	Job efficiency. Includes summary of job-weighted job efficiency percent 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.
NHREQUEST	Node-hours requested. Includes summary of total node-hours requested and total samples.
NHRUN	Node-hours run. Includes summary of total node-hours run and total samples.
QOSDELIVERED	Quality of service delivered. Includes summary of job-weighted quality of service success rate and total samples.
WCACCURACY	Wall clock accuracy. Includes summary of overall wall clock accuracy and total samples.

NOTE

The *STATISTICTYPE* parameter value must be entered in uppercase characters.

Flags

-h Help for this command.

Description

This command displays a table of the selected Maui Scheduler statistics, such as expansion factor, bypass count, jobs, node-hours, wall clock accuracy, and backfill information.

Example

```
% showgrid 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 `showgrid` command returns a table with data for the specified *STASTICTYPE* parameter. The left-most column shows the maximum number of nodes required by the jobs shown in the other columns. The column heads indicate the maximum wall clock 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.

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 nodes. Totals along the bottom provide overall XFactor averages weighted by job, node, and node-seconds.

Related Commands

None.

Default File Location

`/u/loadl/maui/bin/showgrid`

Notes

None.

showq

---

showq [-i | -r] [-p *PARTITION*] [-h]

Purpose

Shows information about running, idle, and non-queued jobs.

Permissions

This command can be run by any user. However, the -i and -r flags can only be used by Maui Scheduler Administrators.

Parameters

PARTITION Partition number that you wish to inspect.

Flags

-h Help for this command.

-i Used by Maui Scheduler Administrators to display idle jobs only.

-p Inspect partition specified with *PARTITION* parameter.

-r Used by Maui Scheduler Administrators to display running jobs only.

Description

Since LoadLeveler is not actually scheduling jobs, the job ordering it displays is no longer valid. The `showq` command displays the actual job ordering under the Maui Scheduler. When used without flags, this command displays all jobs in active, idle, and non-queued states.

Example 1

```
% showq

ACTIVE JOBS--------------------
         JOBNAME USERNAME      STATE  PROC  REMAINING            STARTTIME

   fr28n13.709.0 dsheppar    Running     1    0:55:09  Fri Aug 29 13:27:36
  fr28n07.2303.0 dsheppar    Running     1    0:55:10  Fri Aug 29 13:27:37
  fr17n08.1349.0 dsheppar    Running     1    1:02:29  Fri Aug 29 13:34:56
  fr28n15.4355.0 dsheppar    Running     1    1:03:08  Fri Aug 29 13:35:35
  fr28n05.2098.0 ebylaska    Running    16    1:25:17  Fri Aug 29 11:57:45
  fr28n05.2095.0    kossi    Running     1    1:26:24  Fri Aug 29 03:58:51
   fr28n13.683.0   xztang    Running     8    2:23:01  Thu Aug 28 17:52:08
  fr28n15.4354.0  moorejt    Running    16    3:41:06  Fri Aug 29 12:18:33
  fr17n08.1341.0    mukho    Running     8    3:41:48  Thu Aug 28 18:24:15
  fr17n05.1393.0    zhong    Running     8    4:01:47  Fri Aug 29 04:39:14
  fr28n05.2097.0    zhong    Running     8    4:50:03  Fri Aug 29 05:27:30
  fr28n11.3080.0    mukho    Running     8    5:12:21  Thu Aug 28 19:54:48
   fr28n13.682.0   wengel    Running    32    5:23:51  Thu Aug 28 19:56:58
  fr28n05.2064.0   vertex    Running     1    6:29:55  Thu Aug 28 23:02:22
  fr28n11.3037.0   vertex    Running     1    6:29:55  Thu Aug 28 23:02:22
    fr28n09.26.0    rampi    Running     1    8:37:27  Thu Aug 28 11:09:54
  fr17n08.1328.0   vertex    Running     1    9:29:49  Fri Aug 29 02:02:16
  fr17n10.1467.0    kossi    Running     1   10:27:10  Fri Aug 29 12:59:37
    fr28n09.49.0 holdzkom    Running     8   13:13:08  Fri Aug 29 11:45:35
  fr17n07.1498.0    jpark   Starting    16   14:10:05  Fri Aug 29 04:42:32
  fr17n05.1384.0    zhong    Running     8   18:45:27  Fri Aug 29 14:22:54
  fr28n07.2300.0  jimenez    Running    16   18:54:12  Fri Aug 29 09:26:39
   fr17n09.529.0   vertex    Running     1   19:03:49  Fri Aug 29 11:36:16
  fr28n01.1851.0   vertex    Running     1   19:09:49  Fri Aug 29 11:42:16
  fr17n11.1380.0   vertex    Running     1   19:41:22  Fri Aug 29 12:13:49
  fr17n16.1533.0   vertex    Running     1   20:04:32  Fri Aug 29 12:36:59
  fr17n06.1502.0   vertex    Running     1   20:16:24  Fri Aug 29 12:48:51
```

```
  fr17n10.1466.0   wengel    Running    32   20:24:04  Fri Aug 29 10:58:11
   fr28n13.701.0     kudo    Running     8   20:25:46  Fri Aug 29 10:58:13
  fr28n03.1689.0   vertex    Running     1   20:50:31  Fri Aug 29 13:22:58
   fr28n13.631.0   vertex    Running     1   21:17:40  Fri Aug 29 13:50:07
   fr28n13.708.0     yshi    Running     8   22:49:10  Fri Aug 29 13:21:37
  fr17n05.1395.0     yshi    Running     8   23:36:36  Fri Aug 29 14:09:03
  fr17n11.1388.0 jshoemak    Running    24   23:51:10  Fri Aug 29 14:23:37
  fr28n07.2304.0  rich001    Running     1   26:09:44  Fri Aug 29 13:42:11
  fr28n11.3091.0    rampi    Running     1   26:57:00  Fri Aug 29 05:29:27

36 Active Jobs   251 of 254 Processors Active   (Efficiency:    98.82)

IDLE JOBS----------------------
         JOBNAME USERNAME      STATE  PROC   CPULIMIT            QUEUETIME

  fr28n03.1718.0  ozturan       Idle    64    0:16:40  Thu Aug 28 22:25:48
  fr17n03.1430.0    jason       Idle   128    2:00:00  Wed Aug 27 00:56:49
  fr17n08.1331.0    jason       Idle   128    2:00:00  Wed Aug 27 00:56:21
  fr17n15.1393.0  moraiti       Idle   128    3:20:00  Fri Aug 29 09:58:56
   fr17n09.534.0  kdeacon       Idle    64    1:00:00  Fri Aug 29 04:38:48
   fr28n13.697.0    jpark       Idle    16   24:00:00  Fri Aug 29 03:44:45
  fr17n07.1499.0    jpark       Idle    16   24:00:00  Fri Aug 29 04:42:31
  fr17n06.1517.0   cholik       Idle    16   24:00:00  Fri Aug 29 06:45:46
   fr28n13.706.0  moorejt       Idle    16    5:55:00  Fri Aug 29 10:53:53
  fr17n16.1550.0  moorejt       Idle    16    7:55:00  Fri Aug 29 10:53:54
  fr17n12.1528.0 ebylaska       Idle    16    3:59:59  Fri Aug 29 12:11:30
  fr28n15.4356.0 dsheppar       Idle    16    3:00:00  Fri Aug 29 14:01:42
    fr28n09.50.0 dsheppar       Idle    16    3:00:00  Fri Aug 29 14:01:59
    fr28n09.51.0    zhong       Idle     8   13:55:00  Fri Aug 29 14:07:16
  fr17n16.1551.0    jacob       Idle     4    4:00:00  Fri Aug 29 12:51:19

15 Idle Job(s)

NON-QUEUED JOBS----------------
         JOBNAME USERNAME      STATE  PROC   CPULIMIT            QUEUETIME

  fr17n02.1476.0   vertex       Idle     1   22:00:00  Thu Aug 28 23:48:16
  fr17n05.1392.0   vertex SystemHold     1   22:00:00  Thu Aug 28 23:49:51
  fr17n10.1449.0   vertex       Idle     1   22:00:00  Tue Aug 26 23:49:51
  fr28n03.1674.0    maxia   UserHold     8   23:56:00  Mon Aug 25 16:22:10
  fr28n05.1581.0     sidt   UserHold     1    1:00:00  Sun Jul 27 12:46:17
  fr28n05.2092.0   vertex       Idle     1   22:00:00  Thu Aug 28 23:48:40
   fr28n13.705.2     gigi  NotQueued    32   15:58:00  Fri Aug 29 10:49:01
   fr28n13.705.3     gigi  NotQueued    32   13:58:00  Fri Aug 29 10:49:01
  fr17n08.1349.7 dsheppar  BatchHold     1    2:00:00  Fri Aug 29 13:34:44
  fr28n15.4355.1 dsheppar       Idle     1    2:00:00  Fri Aug 29 13:35:04
  fr28n15.4355.2 dsheppar   Deferred     1    2:00:00  Fri Aug 29 13:35:04
  fr28n15.4355.3 dsheppar       Idle     1    2:00:00  Fri Aug 29 13:35:04

Total Jobs: 63   Active Jobs: 36   Idle Jobs: 15   Non-Queued Jobs: 12
```

The output of this command is divided into three parts, Active Jobs, Idle Jobs, and Non-Queued Jobs.

Active jobs are those that are Running or Starting and consuming CPU resources. Displayed are the job name, 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.

Idle 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 wall clock 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.

Non-Queued 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:

Idle	Job violates a fairness policy. Use `diagnose -q` for more information.
UserHold	A LoadLeveler User Hold is in place.
SystemHold	A LoadLeveler System Hold is in place.

BatchHold	A Maui 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 LoadLeveler has repeatedly failed in attempts to start the job).
Deferred	A Maui 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 LoadLeveler 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 2

```
% showq -r
```

```
           JobName S  Pa Effic  XFactor  Q      User    Group  Nodes  Remaining            StartTime

     fr28n13.709.0 R   1 99.37      1.0  0  dsheppar      daf      1    0:55:50  Fri Aug 29 13:27:36
    fr28n07.2303.0 R   2 98.57      1.0  0  dsheppar      daf      1    0:55:51  Fri Aug 29 13:27:37
    fr17n08.1349.0 R   1 97.94      1.0  0  dsheppar      daf      1    1:03:10  Fri Aug 29 13:34:56
    fr28n15.4355.0 R   3 98.91      1.0  0  dsheppar      daf     64    1:03:49  Fri Aug 29 13:35:35
    fr28n05.2098.0 R   1 94.26      1.3  0  ebylaska    dnavy     16    1:25:58  Fri Aug 29 11:57:45
    fr28n05.2095.0 R   1 98.56      1.0  0     kossi      daf      1    1:27:05  Fri Aug 29 03:58:51
     fr28n13.683.0 R   1 99.75      1.0  0    xztang      daf      8    2:23:42  Thu Aug 28 17:52:08
    fr28n15.4354.0 R   3 98.90      1.2  0   moorejt      daf     16    3:41:47  Fri Aug 29 12:18:33
    fr17n08.1341.0 R   2 99.67      1.0  0     mukho    dnavy      8    3:42:29  Thu Aug 28 18:24:15
    fr28n05.2097.0 R   1 99.70      1.0  0     zhong     govt      8    4:50:44  Fri Aug 29 05:27:30
     fr28n13.682.0 R   3 99.83      1.1  0    wengel     univ     32    5:24:32  Thu Aug 28 19:56:58
    fr17n08.1328.0 R   3 99.69      2.5  0    vertex     univ      1    9:30:30  Fri Aug 29 02:02:16
    fr17n10.1467.0 R   3 98.12      1.0  0     kossi      daf      1   10:27:51  Fri Aug 29 12:59:37
    fr28n07.2300.0 R   1 97.60      1.1  0   jimenez    dnavy     16   18:54:53  Fri Aug 29 09:26:39
     fr17n09.529.0 R   1 99.10      2.9  0    vertex     univ      1   19:04:30  Fri Aug 29 11:36:16
    fr28n01.1851.0 R   1 98.01      2.9  0    vertex     univ      1   19:10:30  Fri Aug 29 11:42:16
    fr17n10.1466.0 R   1 99.51      1.2  0    wengel     univ     32   20:24:45  Fri Aug 29 10:58:11
     fr28n13.701.0 R   3 98.91      1.2  0      kudo      daf      8   20:26:27  Fri Aug 29 10:58:13
     fr28n13.631.0 R   1 99.89      3.0  0    vertex     univ      1   21:18:21  Fri Aug 29 13:50:07
    fr17n05.1395.0 R   2 95.29      1.0  0      yshi     univ      8   23:37:17  Fri Aug 29 14:09:03
    fr17n11.1388.0 R   2 63.46      1.4  0  jshoemak      daf     24   23:51:51  Fri Aug 29 14:23:37
    fr28n07.2304.0 R   1 97.62      1.0  0   rich001      daf      1   26:10:25  Fri Aug 29 13:42:11
    fr28n11.3091.0 R   1 98.87      1.0  0     rampi     univ      1   26:57:41  Fri Aug 29 05:29:27

23 Jobs   251 of 254 Processors Active   (Efficiency:    98.82)
```

The fields are as follows:

JobName	Name of running job.
S	Job State. Either "R" for Running or "S" for Starting.
Pa	Partition in which job is running.
Effic	CPU efficiency of job.
XFactor	Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality Of Service specified for job.
User	User owning job.
Group	Primary group of job owner.
Nodes	Number of processors being used by the job.
Remaining	Time the job has until it has reached its wall clock limit. Time specified in HH:MM:SS notation.
StartTime	Time job started running.

After displaying the running jobs, a summary is provided indicating the number of jobs, the number of allocated processors, and the system utilization.

Example 3

```
% showq -i
```

```
           JobName    Priority  XFactor  Q      User    Group  Nodes   WCLimit     Class
SystemQueueTime

    fr28n03.1718.0*   97615272     59.0  0   ozturan     govt     64   0:16:40     batch  Thu Aug 28
22:25:48
    fr17n03.1430.0      125372     11.0  0     jason      asp    128   2:00:00    medium  Thu Aug 28
18:29:26
     fr28n13.634.0      125365     11.0  0     jason      asp    128   2:00:00    medium  Thu Aug 28
18:30:04
      fr28n09.32.0      118071      7.0  0   moraiti     univ    128   3:20:00     batch  Thu Aug 28
18:32:58
    fr17n15.1393.0      110712      2.4  0   moraiti     univ    128   3:20:00     batch  Fri Aug 29
09:58:56
     fr17n09.534.0       68841     10.9  0   kdeacon      pdc     64   1:00:00     batch  Fri Aug 29
04:38:48
     fr28n13.697.0       21102      1.4  0     jpark    dnavy     16  24:00:00     batch  Fri Aug 29
03:44:45
    fr17n07.1499.0       20906      1.4  0     jpark    dnavy     16  24:00:00     batch  Fri Aug 29
04:42:31
    fr17n06.1517.0       20604      1.3  0    cholik     univ     16  24:00:00     batch  Fri Aug 29
06:45:46
     fr28n13.706.0       20180      1.6  0   moorejt      daf     16   5:55:00     batch  Fri Aug 29
10:53:53
    fr17n16.1550.0       20024      1.5  0   moorejt      daf     16   7:55:00     batch  Fri Aug 29
10:53:54
    fr17n12.1528.0       19916      1.6  0  ebylaska    dnavy     16   3:59:59     batch  Fri Aug 29
12:11:30
      fr28n09.50.0       19097      1.2  0  dsheppar      daf     16   3:00:00     batch  Fri Aug 29
14:01:59
      fr28n09.51.0       12547      1.0  0     zhong     govt      8  13:55:00     batch  Fri Aug 29
14:07:16
    fr17n16.1551.0        9390      1.0  0     jacob     univ      4   4:00:00     batch  Fri Aug 29
14:22:09

Jobs: 15  Total BackLog:  6434 Node Hours  ( 25.33 Hours)
```

The fields are as follows:

JobName	Name of job.
Priority	Calculated job priority.
XFactor	Current expansion factor of job, where XFactor = (QueueTime + WallClockLimit) / WallClockLimit
Q	Quality Of Service specified for job.
User	User owning job.
Group	Primary group of job owner.
Nodes	Minimum number of processors required to run job.
WCLimit	Wall clock limit specified for job. Time specified in HH:MM:SS notation.
Class	Class requested by job.
SystemQueueTime	Time job was admitted into the system queue.

An asterisk at the end of a job (job fr28n03.1718.0* 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.

After displaying the job listing, the command summarizes the workload in the idle queue and indicates the total workload backlog in node-hours. The value in parenthesis indicates the minimum amount of time required to run this workload using the currently available nodes on the system.

Related Commands

Use the `showbf` command to see how many nodes are available for use.

Use the `diagnose` command to show the partitions.

Use the `checkjob` command to check the status of a particular job.

Default File Location

`/u/loadl/maui/bin/showq`

Notes

None.

showres

---

showres [ARGS] [<RESID>]

Purpose: show detailed reservation information

Argument	Description
-g	show '*grep*'-able output with nodename on every line
-h	show usage *help*
-n	display information regarding all *nodes* reserved by <RESID>
-o	display all reservations which *overlap* <RESID> in time
-r	display reservation timeframes in *relative* time mode
-s	display *summary* reservation information
-v	show *verbose* output. If used with the '**-n**' flag, the command will display all reservations found on nodes contained in <RESID>. Otherwise, it will show long reservation start dates including the reservation year.

Parameter	Description
RESID	ID of reservation of interest - optional

Access

This command can be run by any Maui administrator, or by any valid user if the parameter **RESCTLPOLICY** is set to **ANY**.

Description

This command displays all reservations currently in place within the Maui Scheduler. The default behavior is to display reservations on a reservation-by-reservation basis.

Example 1

```
> showres
Reservations

  Type       ReservationID  S       Start         End    Duration  Nodes  StartTime

   Job        fr4n01.902.0  S    -0:02:00     0:08:00     0:10:00     16  Sat Dec 14 08:29:09
   Job        fr5n11.176.0  S    -0:01:00     1:59:00     2:00:00      8  Sat Dec 14 08:30:09
   Job        fr5n11.177.0  S    -0:01:00     0:02:20     0:03:20      1  Sat Dec 14 08:30:09
   Job        fr5n12.179.0  S    -0:00:30     1:59:30     2:00:00      3  Sat Dec 14 08:30:39
   Job        fr5n12.180.0  S    -0:00:30     0:29:30     0:30:00      4  Sat Dec 14 08:30:39
   Job        fr5n13.155.0  S     0:00:00     2:00:00     2:00:00      4  Sat Dec 14 08:31:09
 Group              daf#0   -    10:00:00    INFINITY    INFINITY     16  Sat Dec 14 18:31:09
  User             loadl#0  -     0:00:00    30:00:00    30:00:00     16  Sat Dec 14 08:31:09
System           SYSTEM#0  -    20:00:00    30:00:00    10:00:00     40  Sun Dec 15 04:31:09

25 Reservations Located
```

This example shows all reservations on the system. The fields are as follows:

Type	Reservation Type. This will be one of the following: Job, User, Group, Account, or System.

ReservationID	This is 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. Reservation 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 2

```
> showres -n
Reservations on Sat Dec 14 08:31:09

            NodeName    Type      ReservationID   JobState      Start   Duration
StartTime

   fr10n11.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
   fr26n01.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
    fr5n09.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
                      System           SYSTEM#0        N/A   20:00:00   10:00:00  Sun Dec 15
04:31:09
   fr18n15.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
   fr20n02.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
                       User            loadl#0        N/A    0:00:00   30:00:00  Sat Dec 14
08:31:09
                      Group              daf#0        N/A   10:00:00   INFINITE  Sat Dec 14
18:31:09
   fr20n15.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
                       User            loadl#0        N/A    0:00:00   30:00:00  Sat Dec 14
08:31:09
                      Group              daf#0        N/A   10:00:00   INFINITE  Sat Dec 14
18:31:09
   fr26n11.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
   fr17n11.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
   fr25n12.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
   fr26n16.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
    fr5n12.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
                      System           SYSTEM#0        N/A   20:00:00   10:00:00  Sun Dec 15
04:31:09
    fr5n15.mhpcc.edu     Job       fr4n02.126.0   Starting   -0:02:00    6:00:00  Sat Dec 14
08:29:09
```

This example shows reservations for nodes. The fields are as follows:

NodeName	Node on which reservation is placed.
Type	Reservation Type. This will be one of the following: Job, User, Group, Account, or System.
ReservationID	This is 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.
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 Reservation became active.

Example 3

```
> showres fr35n08.3360.0
Reservations

  Type       ReservationID  S       Start         End    Duration  Nodes  StartTime

   Job      fr35n08.3360.0  S    -8:24:06    15:35:54    24:00:00     16  Thu Mar  5 03:08:38

1 reservation located
```

In this example, information for a specific reservation (job) is displayed.

See Also:

- `setres` - create new reservations.
- `releaseres` - release existing reservations.
- `diagnose -r` - diagnose/view the state of existing reservations.
- Reservation Overview - description of reservations and their use.

---

showstart

---

showstart [-h] <JOBID>

Purpose

Display the earliest possible start and completion times for a specified job.

Permissions

This command can be run by any user.

Parameters

JOBID **Job to be checked**

Flags

-h Help for this command.

Description

This command displays the earliest possible start time of a job. If the job already possesses a reservation, the start time of this reservation will be reported. If no such reservation exists, this command will determine the earliest time a reservation would be created *assuming this job was highest priority*. If this job does not have a reservation and it is not highest priority, the value of returned information may be limited.

Example

```
> showstart job001

job Job001 requires 2 procs for 0:33:20
Earliest start is in          1:40:00 on Thu Jan  1 01:16:40
Earliest Completion is in     2:13:20 on Thu Jan  1 01:50:00
Best Partition: DEFAULT
```

Related Commands

checkjob, **showres**

Notes

Since the information provided by this job is only highly accurate if the job is highest priority or if the job has a reservation, sites wishing to make decisions based on this

information may 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 resulting in slightly lower system utilization (typically less than 8% reduction).

---

showstate

---

showstate [-h]

Purpose

Summarizes the state of the system.

Permissions

This command can be run by any Maui Scheduler Administrator.

Parameters

None.

Flags

-h Shows help for this command.

Description

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. Simple diagnostic tests are also performed and any problems found are reported.

Example

```
% showstate

BQS Summary on Tue May 20 21:18:08 1997


            JobName Nodes  WCLimit   JobState
(A)   fr17n11.942.0    16      600    Running
(B)  fr15n09.1097.0    32    14100   Starting
(C)   fr17n01.942.0     8     6900    Running
(D)    fr13n03.24.0     8    28800   Starting
(E)  fr15n13.1018.0     8    28800   Starting
(F)   fr17n05.953.0     8    86400    Running
(G)  fr15n09.1103.0     1    86340    Running
(H)    fr13n11.28.0     1    86400    Running
(I)    fr14n09.21.0    24    86400   Starting

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
Active Job[ 1]     fr15n09.1097.0 (Starting) Has Node fr7n09 Allocated which is in state 'Idle'
Active Job[ 1]     fr15n09.1097.0 (Starting) Has Node fr15n01 Allocated which is in state 'Idle'
Active Job[ 1]     fr15n09.1097.0 (Starting) Has Node fr15n03 Allocated which is in state 'Idle'
Active Job[ 1]     fr15n09.1097.0 (Starting) Has Node fr15n05 Allocated which is in state 'Idle'
Active Job[ 1]     fr15n09.1097.0 (Starting) Has Node fr15n07 Allocated which is in state 'Idle'
Node fr11n08 is Busy but Has No Job Scheduled
```

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, fr3n06 ...

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 Commands

None.

Default File Location

`/u/loadl/maui/bin/showstate`

Notes

None.

showstats

---

showstats [FLAGS]

Purpose

Show resource usage statistics

Access

This command can be run by any Maui level 1, 2, or 3 Administrator.

Parameters

[NONE]

Flags

NOTE: this command supports all generic maui command flags

Flag	Description
-a [<ACCOUNTID>]	display account statistics
-g [<GROUPID>]	display group statistics
-n [<NODEID>]	display node statistics
-S	display summary information. **NOTE**: only valid with the '**-n**' flag
-s	display general scheduler statistics
-u [<USERID>]	display user statistics
-v	display verbose information

Description

This command shows various accounting statistics for the system. Historical statistics cover the timeframe from the most recent execution of the resetstats command.

Example 1

```
% 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
```

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.

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:

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.
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 wall clock accuracy for jobs completed. Wall clock accuracy is calculated by dividing a job's actual run time by its specified wall clock limit.

* These fields are empty until an account has completed at least one job.

Example 2

```
% 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
```

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.

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:

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.
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 proce-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 wall clock accuracy for jobs completed. Wall clock accuracy is calculated by dividing a job's actual run time by its specified wall clock limit.

* These fields are empty until a group has completed at least one job.

Example 3

```
% showstats -n -S
Memory Requirement Breakdown:

  Memory Nodes Percent InitialNH Percent  NodeHours Percent
      64     8    2.78      9799  794.92       1232  100.00
```

```
   128   144   50.00      9162   41.29      22190  100.00
   256    32   11.11     20290  411.47       4931  100.00
   512    96   33.33      5080   34.34      14793  100.00
  1024     8    2.78        48    3.89       1232  100.00
  2048     0    0.00         0    0.00          0    0.00
 TOTAL   288  100.00     44381  100.00      44381  100.00


Node Statistics

Summary:    8   64MB Nodes   99.26% Avail   79.18% Busy  (Current: 100.00% Avail  100.00% Busy)
Summary:  144  128MB Nodes   98.99% Avail   75.92% Busy  (Current: 100.00% Avail  100.00% Busy)
Summary:   32  256MB Nodes   97.69% Avail   85.66% Busy  (Current: 100.00% Avail  100.00% Busy)
Summary:   96  512MB Nodes   96.12% Avail   82.92% Busy  (Current:  98.96% Avail   94.79% Busy)
Summary:    8 1024MB Nodes   99.87% Avail   81.77% Busy  (Current: 100.00% Avail   75.00% Busy)
System Summary:  288 Nodes   97.92% Avail   79.59% Busy  (Current:  99.65% Avail   97.57% Busy)
```

This example shows a statistical listing of nodes and memory. Memory Requirement Breakdown portion shows information about the current workload profile. In this example, the system monitored is a heterogeneous environment consisting of eight 64 MB (RAM) nodes, 144 128 MB nodes, *etc.*, with a total of 288 nodes. The third column indicates the percentage of total nodes that meet this memory criteria. For example, the eight 64 MB nodes make up 2.78% of the 288 total nodes in the system.

The idle job queue monitored in this example consists of numerous jobs consisting of a total of 44,381 node-hours of work. The node-hour workload of jobs that have specific node memory requirements are assigned to the corresponding memory class. If no specific memory requirement is specified, the job's node-hours are assigned to the lowest memory class, in this case, the 64 MB nodes.

Example 4

```
% showstats

Maui running for       22:01:00  stats initialized on Mon Mar 26 17:43:34

Eligible/Idle Jobs:                   15/45         (33.333%)
Active Jobs:                          42
Successful/Completed Jobs:           873/875        (99.7%)
Avg/Max QTime (Hours):              2.71/4.50
Avg/Max XFactor:                    1.03/4.79

Dedicated/Total ProcHours:       4353.55/4782.10    (91.038%)

Current Active/Total Procs:         183/192         (95.312%)

Avg WallClock Accuracy:            43.25%
Avg Job Proc Efficiency:           98.17%
Est/Avg Backlog (Hours):            34.5/41.8
```

This example shows a concise summary of the system scheduling state. Note that `showstats` and `showstats -s` are equivalent.

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 Maui Scheduler has been scheduling in HH:MM:SS notation followed by the statistics initialization time.

The fields are as follows:

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 wall clock accuracy of completed jobs (job-weighted average).
Job Efficiency	Average job efficiency (UtilizedTime / DedicatedTime).
Est Backlog	Estimated backlog of queued work in hours.
Avg Backlog	Average backlog of queued work in hours.

Example 5

```
% 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
```

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.

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:

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 prochours 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 wall clock accuracy for jobs completed. Wall clock accuracy is calculated by dividing a job's actual run time by its specified wall clock limit.

* These fields are empty until a user has completed at least one job.

Related Commands

Use the `resetstats` command to re-initialize statistics.

Notes

See the Statistics document for more details about scheduler statistics.