Sun™ Grid Engine, Enterprise Edition 5.3 Basics of Administration

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 816-7409-10
July 2002, Revision A

Send comments about this document to: docfeedback@sun.com

Please
Recycle

Contents

Sun Grid Engine, Enterprise Edition 5.3 Basics of Administration

This document is furnished as a quick-reference handbook to aid administrators of Sun™ Grid Engine, Enterprise Edition 5.3 systems. In addition to instructions for common tasks involved in such administration, this document includes a brief discussion of grid computing in general and the Sun Grid Engine, Enterprise Edition 5.3 product specifically. This document concludes with several fine-tuning and troubleshooting tips.

Use this handbook as a supplement to—and not a replacement for—the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User's Guide*, in which you can find detailed explanations of Sun Grid Engine, Enterprise Edition 5.3 concepts and procedures.

Instructions for accomplishing the following tasks are included in this document.

Note – Much of the material in this document appeared originally in the "How-To" section of the Sun Grid Engine project website. Updated frequently, this website is of special value to Sun Grid Engine, Enterprise Edition 5.3 system administrators and is well worthwhile consulting from time to time. The URL for the website is: http://gridengine.sunsource.net/project/gridengine/howto/howto.html

---

About Grid Computing and Sun Grid Engine, Enterprise Edition 5.3 Software

If you are new to *grid* computing administration, you belong to the overwhelming majority of professional computer administrators. Very few people on earth can yet claim to be truly experienced administrators of grid computing environments, because the enabling software technology is less than a decade old. According to Dr. Ian Foster, one of the pioneers in the field of grid computing, concepts relating to grid computing were only being "explored" by 1995. Even the earliest adopters of grid technology, therefor, have less than a decade of experience.

But, regardless of how little experience you may have with grid computing, you have vast experience with using a grid—though a grid of a different sort. Each time you turn an electric switch to "on," you are using the resources of a grid. In most industrialized nations, electricity is provided to homes and businesses through a "power grid" that is composed of a number of independent electric power producers who feed their product into a common resource pool, which is called the grid.

The pooling of resources has benefits for both consumers and producers. A power producer benefits by being able to run its power plant at full capacity, even though consumers in its immediate geographic area may not require as much electricity as the plant can produce. Consumers in a rural area far removed from a city, for example, will not typically require as much power as an ordinary power plant can produce. The power grid arrangement, however, enables the plant to sell its "excess" capacity into the grid, thereby serving consumers in distant areas whose own local power plants may not be able to meet local demand.

The ability to put underutilized resources into production and deliver those resources to meet demand, no matter where that demand may be located, is the defining characteristic of the power grid. And by adopting the “grid” name, the pioneers of the grid computing concept have created an appropriate analogy. The fundamental goal of grid computing is to pool the underutilized resources of hundreds—even thousands—of idle and nearly idle computers and make that combined, extreme computing power available to meet the demand of compute-intensive consumers.

The Role of Distributed Resource Management

Where the analogy breaks down somewhat is in the details of delivery. The power grid manages to pool resources by literally wiring together all of the producer members of the grid and, ultimately, the homes, factories, farms, and offices of the electricity consumers. Once connected and fed into the power grid, the electricity flows according to demand. Turn the light switch to “on,” and the electricity flows from a power plant somewhere, through the power grid, and into the light bulb. Computing resources within a grid computing environment can also be said to be “wired” together, although by networking technology: the Internet, for the primary example, but also by intranets. But you can’t simply turn on a switch on a network computer and expect to use the resources of (or contribute resources to) a computing grid.

Access to a computing grid is enabled only after you have installed special *middleware* software. Middleware is a software layer that lies between the computer operating system, such as Sun’s Solaris™ Operating Environment or the Linux operating system, and application software, such as a three-dimensional graphics rendering program. Sun Grid Engine software is middleware that enables participation in computing grids.

The tangible result of an early “open-source” grid computing project, Sun Grid Engine is *distributed resource management* software that mediates between multiple users and multiple computational resources—often located remotely—to provide users with vastly more computational power for complex jobs. Through Sun Grid Engine software, idle computing resources joined to the grid become available to often distant users, thereby increasing the productivity of both the users and the resources. For example, computing resources that had ordinarily run at a quarter of their actual capacity have been shown to be running at nearly 100 percent through participation in a grid.

The computational devices account for the “distributed resource” portion of the term, “distributed resource management.” Sun Grid Engine software accounts for the indispensable portion of the term, “management.” Without such management, chaos would result. As the grid manager, Sun Grid Engine software accepts jobs

submitted by users and schedules them for execution on appropriate systems in the grid based upon resource management *policies*, which are set by the organization's technical and management staff.

No two grids are alike; one size does not fit all situations. There are three key classes of grids, which scale from single systems to supercomputer-class compute farms that utilize thousands of processors:

- *Cluster grids*, enabled by the basic Sun Grid Engine 5.3 software, are the simplest, consisting of computer *hosts* working together to provide a single point of access to users in a single project or department.

  Sun Grid Engine, Enterprise Edition 5.3 software extends this basic model to create the other two, more complex and powerful, classes of grids.

- *Campus grids* enable multiple projects or departments within an organization to share computing resources. Organizations can use campus grids to handle a wide variety of tasks, from cyclical business processes to rendering, data mining, and more.

- *Global grids* are a collection of campus grids that cross organizational boundaries to create very large virtual systems. Users have access to compute power that far exceeds the resources available within their own organization.

Sun Grid Engine, Enterprise Edition 5.3 software provides the power and flexibility required for Campus grids. The product is very useful for existing cluster grids enabled by the basic Sun Grid Engine software, as it facilitates a smooth transition to create a Campus grid by consolidating all existing Sun Grid Engine cluster grids on the campus. The product is also a good start for an enterprise campus that makes the move to the grid computing model for the first time.

Policies, Hosts, and Daemons

Sun Grid Engine, Enterprise Edition 5.3 software orchestrates the delivery of computational power based upon enterprise resource policies that you, the system administrator, maintain. The Sun Grid Engine, Enterprise Edition 5.3 system uses these policies to examine the available computational resources within the Campus grid, gathers these resources, and then allocates and delivers them automatically in a way that optimizes usage across the Campus grid.

To enable cooperation within the Campus grid, project owners using the grid need to negotiate policies, have flexibility in the policies for manual overrides for unique project requirements, and have the policies automatically monitored and enforced.

As administrator, you can define high-level utilization policies, customized according to whatever is appropriate for your site. Four such policies, described in detail in the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User's Guide*, are available.

- Functional
- Share-based
- Deadline
- Override

Sun Grid Engine, Enterprise Edition 5.3 policy management automatically controls the use of shared resources in the cluster to achieve your goals. High-priority jobs are dispatched preferentially and receive greater CPU entitlements when they are competing with other, lower-priority, jobs. Sun Grid Engine, Enterprise Edition 5.3 software monitors the progress of all jobs and adjusts their relative priorities correspondingly, and with respect to the goals that you defined in the policies.

Whether your site's usage rules are loose or strict, the Sun Grid Engine, Enterprise Edition 5.3 policy module accommodates them. This policy-based resource allocation grants each user, team, department, and all projects an allocated share of system resources during an accumulation period such as a week, month, or quarter.

Four types of hosts are fundamental to the Sun Grid Engine, Enterprise Edition 5.3 system.

- Master
- Execution
- Administration
- Submit

In addition, it is advisable that you, as administrator, create a *shadow master host*. While there is only one master host, other machines in the cluster can be designated as shadow master hosts to provide greater availability. A shadow master host continually monitors the master host, and automatically and transparently assumes control in the event that the master host fails. Jobs already in the cluster are not affected by a master host failure.

Each host type is described in detail in the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User's Guide*.

Four daemons provide Sun Grid Engine, Enterprise Edition 5.3 functionality.

- `sge_qmaster` is the master daemon.
- `sge_schedd` is the scheduler daemon.
- `sge_execd` is the execution daemon.
- `sge_commd` is the communication daemon.

Each of these daemons is discussed in detail in the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User's Guide*.

The remainder of this document provides instructions on how to accomplish the most common Sun Grid Engine, Enterprise Edition 5.3 administrative tasks. For additional background information and for instructions on how to accomplish tasks

not presented here, see the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User's Guide*, the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual*, or the `man` pages.

Command-Line Instructions for Common Administration Tasks

Note – The following groups of instructions are based on using commands at the command line only. All of the commands have their counterpart functionality in the Sun Grid Engine, Enterprise Edition 5.3 graphical user interface, `QMON`, as well. For instructions on how to use `QMON` to accomplish the following tasks, see the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User's Guide*.

▼ How To Add or Remove Administrative Privileges from a Host

- **To add administrative privileges, enter the following command.**

```
qconf -ah
```

- **To remove administrative privileges, enter the following command.**

```
qconf -dh
```

▼ How To Add an Execution Host

1. **Make the new host an administrative host by entering the following command.**

```
qconf -ah
```

2. **As root on this new host, run the following script from** `$SGE_ROOT`.

```
install_execd
```

▼ How To Remove an Execution Host

- **Delete the queues associated with this host by entering the following command.**

```
qconf -dq
```

- **Delete the host by entering the following command.**

```
qconf -de
```

▼ How To Add or Remove a Submit Host

- **To designate a host as a submit host, enter the following command at the host's command line.**

```
qconf -as
```

- **To remove a host's designation as a submit host, enter the following command at the host's command line.**

```
qconf -ds
```

▼ How To Display the Names of Current Hosts

- **To display the names of administrative hosts, enter the following command.**

```
qconf -sh
```

- **To display the names of submit hosts, enter the following command.**

```
qconf -ss
```

- **To display the names of execution hosts, enter the following command.**

```
qconf -sel
```

▼ How To Administer Queues

- **To add a queue, enter the following command.**

```
qconf -aq
```

- **To add a queue *from file,* enter the following command.**

```
qconf -Aq
```

- **To delete a queue, enter the following command.**

```
qconf -dq
```

- **To modify a queue, enter the following command.**

```
qconf -mq
```

- **To change single attributes of more than one queue, enter the following command.**

```
qconf -mqattr
```

▼ How To Change the Master Host

1. **Stop the master and scheduler daemons on the current master host by entering the following command.**

```
qconf -ks -km
```

2. **Edit the** `$SGE_ROOT/default/common/act_qmaster` **file according to the following guidelines.**

   a. **In the** `act_qmaster` **file, replace the current host name with the new master host's name.**

      This name should be the same as the name returned by the `gethostname` utility. To obtain that name, enter the following command on the new master host.

```
$SGE_ROOT/utilbin/$ARCH/gethostname
```

   b. **Replace the old name in the** `act_qmaster` **file with the name returned by the** `gethostname` **utility.**

3. **On the new master host, execute the following script.**

```
$SGE_ROOT/default/common/sge5
```

   This will start up `sge_qmaster` and `sge_schedd` on the new master host.

▼ How To Set Up a Shadow Master

1. **Create a** `shadow_masters` **file according to the following guidelines.**
   - Create the file in `$SGE_ROOT/default/common`.

- Place the name of the primary master host as the first line, then list the other hosts that are chosen to assume master responsibility in the order you desire. For example:

```
system% cat shadow_masters
host1
host2
host3
```

In the example above, `host1` is the primary master host. Should `host1` fail, `host2` will take over as the master server after a period of approximately 10 minutes. Further, if `host2` should then fail, `host3` will take over.

2. **Verify correct permissions.**

   All master shadow hosts must have read/write permissions to the `qmaster` spool directory.

3. **Start the shadow daemons by using the `sge5` startup script. As root on each host, enter the following command.**

```
$SGE_ROOT/default/common/sge5 -shadowd
```

After you have completed these steps, master shadowing for the Sun Grid Engine, Enterprise Edition 5.3 cluster is active.

Using Scripts and Files for Administrative Tasks

Note – You can use the `QMON` graphical user interface to perform all Sun Grid Engine, Enterprise Edition 5.3 administrative tasks, and using it brings you the added benefit of learning about all of the system's capabilities. However, you can also administer a Sun Grid Engine, Enterprise Edition 5.3 grid through commands issued at the shell prompt and called from within shell scripts. Many experienced administrators find this to be a more flexible, quicker, and more powerful way to change settings. The following three sets of instructions deal with such methods.

▼ How To Add or Modify Objects by Using Files

- **Use the `qconf` command to add or modify objects according to the specifications you create in a file.**

The command has the following syntax: `qconf -{A,M}<object> <filename>`

In the syntax above, `-A` means add and `-M` means modify. The *<object>* variable can be any of the following.

- `c` – Complex
- `ckpt` – Checkpoint environment
- `e` – Execution host
- `p` – Parallel environment
- `q` – Queue
- `u` – User set

You can use this option in combination with the *show* option of the `qconf` command—`qconf -s<obj>`—to take an existing object, modify it, and then update the existing object or create a new one.

Example

The following is an example of a shell script that modifies the *migration command* of an existing checkpoint environment.

```
#!/bin/sh
# ckptmod.sh: modify the migration command
# of a checkpointing environment
# Usage: ckptmod.sh <checkpoint-env-name> <full-path-to-
command>
TMPFILE=/tmp/ckptmod.$$

CKPT=$1
MIGMETHOD=$2

qconf -sckpt $CKPT | grep -v '^migr_command' > $TMPFILE
echo "migr_command $MIGMETHOD" >> $TMPFILE
qconf -Mckpt $TMPFILE
rm $TMPFILE
```

▼ How To Modify Queues, Hosts, and Environments by Using Files

You can modify individual queues, hosts, and both parallel and checkpointing environments from the command line by using the `qconf` command in one of two distinct ways, and in combination with other commands, depending on the arguments to the command. The following instructions cover both.

- **If you have *already prepared a file,* enter the `qconf` command and appropriate options.**

  The command has the following syntax; pay particular attention to the case of the letter options, as the difference between uppercase and lowercase is significant.

  `qconf -M{q,e,p,ckpt}` <*filename*>

  The options have the following meanings.

  - `-M` – Modify from an existing file, <*filename*>
  - `q` – Queue
  - `e` – Execution host
  - `p` – Parallel environment
  - `ckpt` – Checkpoint environment

- **If you have *not already prepared a file,* enter the `qconf` command and appropriate options.**

  The command has the following syntax; pay particular attention to the case of the letter options, as the difference between uppercase and lowercase is significant.

  `qconf -m{q,e,p,ckpt}` <*objname*>

  The options have the following meanings.

  - `-m` – Modify by opening a text editor to create a modification file
  - `q` – Queue
  - `e` – Execution host
  - `p` – Parallel environment
  - `ckpt` – Checkpoint environment

Discussion

The difference between the commands—an uppercase `M` in the first and a lowercase `m` in the second—controls the command's result. Both the `-M` and `-m` options mean *modify*, but the uppercase `M` implies modification from an *existing* file, while the lowercase `m` does not. Instead, the lowercase `-m` opens a temporary file in an editor, and when you save any changes you make to this file and exit the editor, the system immediately reflects those changes.

However, when you want to change many objects at once, or you want to change object configuration non-interactively, use the `qconf -...attr` set of commands.

One such set of commands makes modifications according to specifications *in a file*:

```
qconf -{A,M,R,D}attr queue|exechost|pe|ckpt <filename>
```

A counterpart set of commands makes modifications according to the specification *on the command line*:

```
qconf -{a,m,r,d}attr queue|exechost|pe|ckpt <attrib> <value> <queue_list>|<host_list>
```

In both sets of commands, the options indicate the following:

- `-A/a` – Add attribute
- `-M/m` – Modify attribute
- `-R/r` – Replace attribute
- `-D/d` – Delete attribute
- <*attrib*> – Queue or host attribute to be changed
- <*value*> – Value of attribute to be affected
- <*filename*> – File containing attribute-value pairs

The `a`, `m`, and `d` options enable you to operate on individual values in a list of values, while `r` will replace the entire list of values with the new one which is specified, either on the command line or in the file.

Examples

- **Change the queue type of `tcf27-e019.q` to batch-only:**

```
% qconf -rattr queue qtype batch tcf27-e019.q
```

- **Modify the queue type and shell start behavior of `tcf27-e019.q` based on the contents of the file, `new.cfg`:**

```
% cat new.cfg
qtype batch interactive checkpointing
shell_start_mode unix_behavior
% qconf -Rattr queue new.cfg tcf27-e019.q
```

- **Attach the complexes named `storage` and `license` to the host, `tcf27-e019`:**

```
% qconf -rattr exechost complex_list storage,license tcf27-e019
```

- **Add the resource named** `scratch1` **with a value of 1000M and** `long` **with a value of 2:**

```
% qconf -rattr exechost complex_values scratch1=1000M,long=2 tcf27-e019
```

- **Attach the resource named** `short` **to the host with a value of 4**

```
% qconf -aattr exechost complex_values short=4 tcf27-e019
```

- **Change the value of** `scratch1` **to 500M while leaving other values untouched:**

```
% qconf -mattr exechost complex_values scratch1=500M tcf27-e019
```

- **Delete the resource,** `long`**:**

```
% qconf -dattr exechost complex_values long tcf27-e019
```

- **Add** `tcf27-b011.q` **to the list of queues for the checkpointing environment,** `sph`**:**

```
% qconf -aattr ckpt queue_list tcf27-b011.q sph
```

- **Change the number of slots in the parallel environment,** `make`**, to 50:**

```
% qconf -mattr pe slots 50 make
```

Targeting Queues with the `qselect` Command

The `qselect` command outputs a list of queues. If called with options, it lists only queues that match the given specifications. You can use this command to great advantage in combination with the `qconf -...attr` command set, targeting the specific queues that you want to modify.

Examples

- **To list all queues on Linux machines:**

```
% qselect -l arch=glinux
```

- **To list all queues on machines with two CPUs:**

```
% qselect -l num_proc=2
```

- **To list all queues on all four-CPU 64-bit Solaris machines:**

```
% qselect -l arch=solaris64,num_proc=4
```

- **To list queues that provide an application license (previously configured):**

```
% qselect -l app_lic=TRUE
```

You can combine `qselect` with `qconf` to do wide-reaching changes with a single command line. To do this, put the entire `qselect` command within back slashes (/) and use it in place of the <*queue_list*> variable on the `qconf` command line.

Examples

- **Set the `prolog` script to `sol_prolog.sh` on all queues on Solaris machines:**

```
% qconf -mattr queue prolog /usr/local/scripts/sol_prolog.sh ‘qselect -l
arch=solaris‘
```

- **Set the attribute, `fluent_license`, to two on all queues on two-processor systems:**

```
% qconf -mattr queue complex_values fluent_license=2 ‘qselect -l num_proc=2‘
```

Using the `qconf` command in conjunction with the `qselect` command provides the most flexible way to automate the configuration of Sun Grid Engine, Enterprise Edition 5.3 queues, allowing you to build up your own custom administration scripts.

▼ How To Modify a Global Configuration or Scheduler by Using Files

- **To change a global configuration, use the `qconf -mconf` command, according to guidance in the following section.**
- **To change the scheduler, use the `qconf -msconf` command, according to the guidelines in the following section.**

Discussion

Both of these commands open up a temporary file in an editor. When you exit the editor, any changes you have saved to this temporary file are processed by the system and take effect immediately. The editor program used to open the temporary file is the one specified by the `EDITOR` environment variable. If this variable is undefined, then `vi` is used.

You can take advantage of the `EDITOR` environment variable to automate the behavior of the `qconf -m...` commands. Change the value of this variable to point to a program that modifies the file whose name is given by the first argument. After this program modifies the temporary file and exits, the system will read in the modifications and update immediately.

Note – If the modification time of the file does not change after the edit operation, the system will sometimes incorrectly assume it has not been modified. Therefore, you should insert a `sleep 1` instruction before writing the file, to ensure a different modification time.

Example

The script in the following example modifies the schedule interval of the scheduler.

```
#!/bin/ksh
# sched_int.sh: modify the schedule interval
# usage: sched_int.sh <n>, where <n> is
# the new interval, in seconds. n < 60

TMPFILE=/tmp/sched_int.$$
if [ $MOD_SGE_SCHED_INT ]; then
     grep -v schedule_interval $1 > $TMPFILE
     echo "schedule_interval 0:0:$MOD_SGE_SCHED_INT" >> $TMPFILE
# sleep to ensure modification time changes
     sleep 1
     mv $TMPFILE $1
else
     export EDITOR=$0
     export MOD_SGE_SCHED_INT=$1
     qconf -msconf
fi
```

The script above modifies the `EDITOR` environment to point to itself, and then calls the `qconf -msconf` command. This second nested invocation of the script then modifies the temporary file specified by the first argument, and then exits. The Sun Grid Engine, Enterprise Edition 5.3 system then automatically reads in the changes and the first invocation of the script terminates. The above technique can be used in conjunction with any `qconf -m`... command. However, it is especially useful for administration of the scheduler and global configuration, as there is no other way to automate this.

▼ How To Fine Tune Your Grid Environment

Sun Grid Engine, Enterprise Edition 5.3 is a full function, general purpose Distributed Resource Management (DRM) tool. The scheduler component in the system supports a wide range of different compute farm scenarios. To get the maximum performance from your compute environment, it can be worthwhile to review which features are enabled and which are really needed to solve your load management problem. Disabling some of these features can have a performance benefit on the throughput of your cluster.

Scheduler Monitoring

Scheduler monitoring can help you to find out the reason why certain jobs are not dispatched. However, providing this information for all jobs at any time can be resource consuming and is usually not needed.

- **To disable scheduler monitoring, set** `schedd_job_info` **to false in scheduler configuration,** `sched_conf(5)`**.**

Finished Jobs

In the case of array jobs, the finished job list in `qmaster` can become quite large. Switching it off will both save memory and speed up the `qstat` process, because the `qstat` process also fetches the finished jobs list.

- **To turn off the finished job list function, set** `finished_jobs` **to** `0` **in global configuration,** `sge_conf(5)`**.**

Job Verification

Forcing validation at job submission time can be a valuable procedure to prevent non-dispatchable jobs from forever remaining in a pending state. However, It can also be a time-consuming task to validate jobs, especially in heterogeneous environments with a variety of different execution nodes and consumable resources, and every user having his or her own job profile. In homogenous environments with only a couple of different jobs, a general job validation usually can be omitted.

- **To disable job verification, add the** `qsub(1)` **option,** `-wn`**, in the cluster-wide default requests (see** `sge_request(5)`**).**

Load Thresholds and Suspend Thresholds

Load thresholds are needed if you deliberately oversubscribe your machines and you need a mechanism to prevent excessive system load. Suspend thresholds are also used for this. The other case in which load thresholds are needed is when the execution node is still open for interactive load which is not under control of the Sun Grid Engine, Enterprise Edition 5.3 system and you want to prevent the node from being overloaded.

If a compute farm is more single-purpose—for example, each CPU at a compute node is represented by only one queue slot and no interactive load is expected at these nodes—then you can omit `load_thresholds`.

- **To disable both thresholds, set** `load_thresholds` **to** `none` **and** `suspend_thresholds` **to** `none` **(see** `queue_conf(5)`**).**

Load Adjustments

Load adjustments are used to increase, virtually, the measured load after a job has been dispatched. This mechanism is helpful in case of oversubscribed machines in order to align with load thresholds. You should switch off load adjustments if they are not needed because they impose on the scheduler some additional work in connection with sorting hosts and load thresholds verification.

- **To disable load adjustments, set** `job_load_adjustments` **to** `none` **and** `load_adjustment_decay_time` **to** `0` **in the scheduler configuration,** `sched_conf(5)`**.**

Scheduling-on-Demand

The default for the Sun Grid Engine, Enterprise Edition 5.3 system is to start scheduling runs in a fixed schedule interval (see `schedule_interval` in `schedd_conf(5)`). The good feature about fixed intervals is that they limit the cpu time consumption of the `qmaster`/`scheduler`. The bad feature is that it throttles the scheduler, artificially resulting in a limited throughput. Many compute farms have machines specifically dedicated to `qmaster`/`scheduler` and, in such setups, there is no reason for throttling the scheduler.

- **Configure scheduling-on-demand by using the** `FLUSH_SUBMIT_SEC` **and** `FLUSH_FINISH_SEC` **settings in the** `schedd_params` **section of the global cluster configuration,** `sge_conf(5)`**.**

  If scheduling-on-demand is activated, the throughput of a compute farm is only limited by the power of the machine that is hosting `qmaster`/`scheduler`.

Policies

One of the most important features of Sun Grid Engine, Enterprise Edition 5.3 software is its notion of distributing computational power by way of policies that are set by the system administrator. See the section, “Policies, Hosts, and Daemons” on page 4 for background information.

The following section includes instructions on how to create basic “Functional” and “Share-based” policy setups for your site. Instructions for “Deadline” and “Override” policy setups, as well as for additional Functional and Share-based setups, are included in the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User’s Guide*.

Functional Policy Setups

The setups in this section all use the Functional policy as the main policy. This type of setup ensures that a defined share is guaranteed to each user, project, or department at any time. Jobs of users, projects, or departments that have occupied fewer resources than supposed are preferred when the system dispatches jobs to idle resources.

At the same time, full resource utilization is guaranteed, because unused share proportions are distributed among those users, projects, and departments that need the resources. Past resource consumption is not taken into account.

▼ How To Create User-Based Functional Scheduling

The objective of this setup is to create a certain share assignment of all the resources combined in the Sun Grid Engine, Enterprise Edition 5.3 cluster to different users. FCFS scheduling is used among jobs of the same user.

1. **In the** `schedd_params` **section of global configuration (**`sge_conf(5)`**), use** `SHARE_FUNCTIONAL_SHARES=1`**.**
2. **Specify the number of functional tickets (for example.** `1000000`**) in the Scheduler configuration (**`sched_conf(5)`**).**
3. **Add one user (**`user(5)`**) for each scheduling-relevant user.**
4. **Assign functional shares to each user.**

   Assign the shares as a percentage of the whole. For example:

   - userA (10)
   - userB (20)
   - userC (20)
   - userD (50)

▼ How To Create Project-Based Functional Scheduling

The objective of this setup is to create a certain share assignment of all the resources combined in the Sun Grid Engine, Enterprise Edition 5.3 cluster to different projects. FCFS scheduling is used among jobs of the same project.

1. **In the** `schedd_params` **section of the global configuration (**`sge_conf(5)`**), use** `SHARE_FUNCTIONAL_SHARES=1`**.**
2. **Specify the number of functional tickets (for example.** `1000000`**) in the Scheduler configuration (**`sched_conf(5)`**).**
3. **Add one project (**`project(5)`**) for each scheduling-relevant project.**

   You can control access to higher privileged projects by way of the acl and xacl functions. See the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User's Guide* or the *Sun Grid Engine 5.3 and Sun Grid Engine, Enterprise Edition 5.3 Reference Manual* for information on these functions.

4. **Assign functional shares to each project.**

   Assign the shares as a percentage of the whole. For example:

   - ProjectA (55)
   - ProjectB (45)

▼ How To Create Department-Based Functional Scheduling

The objective of this setup is to create a certain share assignment of all the resources combined in the Sun Grid Engine, Enterprise Edition 5.3 cluster to different departments. FCFS scheduling is used among jobs of the same department.

1. **In the** `schedd_params` **section of the global configuration (**`sge_conf(5)`**), use** `SHARE_FUNCTIONAL_SHARES=1`**.**
2. **Specify the number of functional tickets (for example.** `1000000`**) in the Scheduler configuration (**`sched_conf(5)`**).**
3. **Add each scheduling-relevant department.**
4. **Assign functional shares to each department.**

   Assign the shares as a percentage of the whole. For example:

   - DepartmentA (90)
   - DepartmentB (5)
   - DepartmentC (5)

   The setups in this section all use the Share-tree policy as the main policy. This type of setup ensures that a defined share is guaranteed to the instances configured in the share-tree over time. Jobs associated with share-tree "branches" where fewer resources were consumed in the past than supposed (share-tree share) are preferred

when the system dispatches jobs to idle resources. At the same time, full resource utilization is guaranteed, because unused share proportions are still available for pending jobs associated with other share-tree branches.

▼ How To Create Project-Based Share-Tree Scheduling with FCFS Within Each Project

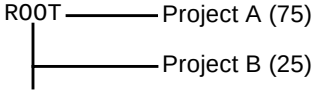
The objective of this setup is to guarantee over time a certain share assignment of all the resources combined in the Sun Grid Engine, Enterprise Edition 5.3 cluster to different projects. FCFS scheduling is used among jobs of the same project.

Note – Some of the instructions in this section—unlike instructions in other sections of this document—require the use of the qmon graphical user interface. See the *Sun Grid Engine, Enterprise Edition 5.3 Administration and User's Guide* for background information about, and instructions for, using qmon.

---

1. **Specify the number of share-tree tickets (for example.** `1000000`**) in the Scheduler configuration (**`sched_conf(5)`**).**
2. **Add one project (**`project(5)`**) for each scheduling-relevant project.**
3. **Using the** `qmon` **graphical user interface, set up a share tree that reflects the structure of all scheduling-relevant projects as nodes.**
4. **Assign share tree shares to the projects.**

   A simple structure would resemble the following example.



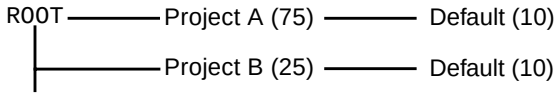
▼ How To Create Project-Based Share-Tree Scheduling with Equal Shares for Each User

The objective of this setup is to guarantee over time a certain share assignment of all the resources combined in the Sun Grid Engine, Enterprise Edition 5.3 cluster to different projects. An equal share is targeted among the jobs competing for resources within the same project.

1. **Specify the number of share-tree tickets (for example.** `1000000`**) in the Scheduler configuration (**`sched_conf(5)`**).**

2. **Add one user (`user(5)`) for each scheduling-relevant user.**
3. **Add one project (`project(5)`) for each scheduling-relevant project.**
4. **Using the `qmon` graphical user interface, set up a share tree that reflects the structure of all scheduling-relevant projects as nodes.**
5. **Assign share tree shares to the projects.**

   A simple structure would resemble the following example.



6. **Add the user `default` as a leaf in the tree below each of these projects**

▼ How To Create Project-Based Share-Tree Scheduling with Per-User Individual Shares Within Each Project

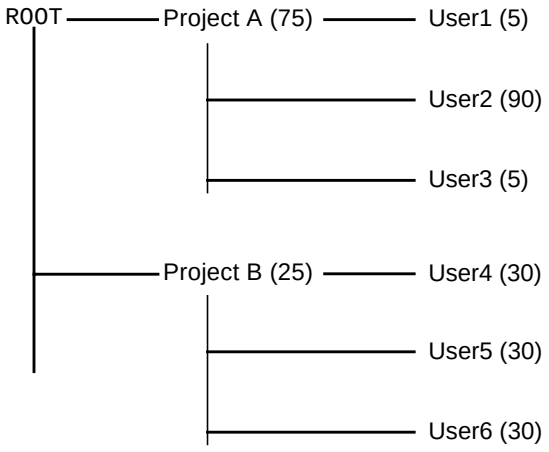
The objective of this setup is to guarantee over time a certain share assignment of all the resources combined in the Sun Grid Engine, Enterprise Edition 5.3 cluster to different projects. Individual share assignments per user is needed.

1. **Specify the number of share-tree tickets (for example. `1000000`) in the Scheduler configuration (`sched_conf(5)`).**
2. **Add one user (`user(5)`) for each scheduling-relevant user.**
3. **Add one project (`project(5)`) for each scheduling-relevant project.**
4. **Using the `qmon` graphical user interface, set up a share tree that reflects the structure of all scheduling-relevant projects as nodes.**

5. **Assign share tree shares to the projects.**

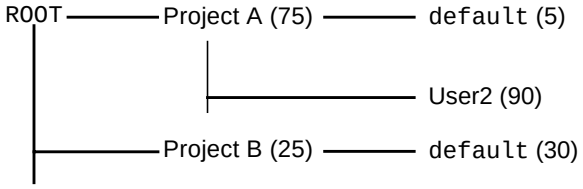
   Add each user as a leaf to the projects where he or she has access and assign individual shares.

   A simple structure would resemble the following example.



Discussion

If individual share assignment is required for just a few users, you can designate the user `default` in combination with individual users below a project node. For example, the tree illustrated above can be condensed into the following.



Troubleshooting Common Problems

Use the following section to help you diagnose and respond to the cause of common problems.

- **Problem** – The output file for your job says, `Warning: no access to tty; thus no job control in this shell....`

- **Possible cause** – One or more of your login files contain an `stty` command. These commands are only useful if there is a terminal present.
- **Possible solution** – In Sun Grid Engine, Enterprise Edition 5.3 batch jobs, there is no terminal associated with these jobs. You must either remove all `stty` commands from your login files, or bracket them with an `if` statement that checks for a terminal before processing. The following is an example of this.

```
/bin/csh:
stty -g            # checks terminal status
if ($status == 0)  # succeeds if a terminal is present
<place all stty commands in here>
endif
```

- **Problem** – The job standard error log file says: `'tty': Ambiguous`. However, there is no reference to `tty` in the user's shell, which is called in the job script.
  - **Possible cause** – `shell_start_mode` is, by default, `posix_compliant`; therefore, all job scripts run with the shell specified in the queue definition, not the one specified on the first line of the job script.
  - **Possible solution** – Use the `-S` flag to the `qsub` command, or change `shell_start_mode` to `unix_behavior`.
- **Problem** – You can run your job script from the command line, but it fails when you run it via the `qsub` command.
  - **Possible cause** – It is possible that process limits are being set for your job. To test this, write a test script that performs `limit` and `limit -h` functions. Execute both interactively at the shell prompt and through the `qsub` command to compare the result.
  - **Possible solution** – Make sure to remove any commands in configuration files that sets limits in your shell.
- **Problem** – Execution hosts report a load of `99.99`.
  - **Possible cause** – Three possibilities exist.

1. The `execd` daemon is not running on the host.
2. A default domain is incorrectly specified.
3. The `qmaster` host sees the name of the execution host as different from the name that the execution host sees for itself.
   - **Possible solution** – Depending on the cause, one of the following solutions may work. (Match the number of the "possible cause" to the number of the following solutions.)

1. As `root`, start up the `execd` daemon on the execution host by running the `$SGE_ROOT/default/common/`'rcsge' script.

2. As the Sun Grid Engine, Enterprise Edition administrator, run the `qconf -mconf` command and change the `default_domain` variable to `none`.

3. If you *are* using DNS for resolving the host names of your compute cluster, configure `/etc/hosts` and NIS to return the fully qualified domain name (FQDN) as the primary host name. (Of course, you may still define and use the short alias name; for example: `168.0.0.1 myhost.dom.com myhost`

   If you *are not* using DNS, make sure that all of your `/etc/hosts` files and your NIS table are consistent; for example: `168.0.0.1 myhost.corp myhost` or `168.0.0.1 myhost`

- **Problem** – Every 30 seconds, a warning similar to the following is printed to *<cell>*`/spool/<host>/messages`:

```
Tue Jan 23 21:20:46 2001|execd|meta|W|local
configuration meta not defined - using global configuration
```

But there *is* a file for each host in *<cell>*`/common/local_conf/`, each with FDQN.

  - **Possible cause** – The host name resolving at your machine, `meta`, returns the short name; but at your master machine, `meta` with FQDN is returned.
  - **Possible solution** – Make sure that all of your `/etc/hosts` files and your NIS table are consistent in this respect. In this example, there could erroneously be a line such as the following in the `/etc/hosts` file of the host, `meta`:

    `168.0.0.1 meta meta.your.domain`

    But the line *should* instead be:

    `168.0.0.1 meta.your.domain meta`.

- **Problem** – Occasionally you see `CHECKSUM ERROR`, `WRITE ERROR`, or `READ ERROR` messages in the `messages` files of the daemons.
  - **Possible cause** – As long as these messages do not appear in a one-second interval (they typically may appear between one and 30 times per day), there is no need to do anything about this issue.
- **Problem** – Jobs will finish on a particular queue, and return the following in `qmaster/messages`:

```
Wed Mar 28 10:57:15 2001|qmaster|masterhost|I|job 490.1
finished on host exechost
```

But then you see the following error messages on the execution host's `exechost/messages` file:

```
Wed Mar 28 10:57:15 2001|execd|exechost|E|can't find directory
"active_jobs/490.1" for reaping job 490.1
```

```
Wed Mar 28 10:57:15 2001|execd|exechost|E|can't remove
directory
"active_jobs/490.1": opendir(active_jobs/490.1) failed:
Input/output error
```

- **Possible cause** – The `$SGE_ROOT` directory, which is automounted, is being unmounted, causing the `sge_execd` daemon to lose its `cwd`.
- **Possible solution** – Use a local spool directory for your `execd` host. Set the parameter, `execd_spool_dir`, using `qmon` or `qconf` commands.

- **Problem** – When submitting interactive jobs with the `qrsh` utility, you receive the following error message:

```
% qrsh -l mem_free=1G error: error: no suitable queues
```

Yet queues are available for batch jobs using the `qsub` utility, and can be queried by using `qhost -l mem_free=1G` and `qstat -f -l mem_free=1G`.

- **Possible cause** – The message, `error: no suitable queues`, results from the `-w e submit` option, which is active by default for interactive jobs such as `qrsh` (look for `-w e` in `qrsh(1)`). This option causes the `submit` command to fail if the `qmaster` does not know for sure that the job will be dispatchable according to the current cluster configuration. The intension of this mechanism is to decline job requests in advance in case they can't be granted.
- **Possible solution** – In this case, `mem_free` is configured to be a consumable resource, but you have not specified the amount of memory that is to be available at each the host. The memory load values are deliberately not considered for this check, because they vary, so they can't be seen as part of the cluster configuration. To overcome this, you can do one of the following.

  Omit this check generally by overriding the `qrsh` default setting, `-w e`, explicitly by submitting it with `-w n`. You can also put this into `$SGE_ROOT/`<*cell*>`/common/cod_request`.

  If you intend to manage `mem_free` as a consumable resource, specify the `mem_free` capacity for your hosts in `complex_values` of `host_conf(5)` by using `qconf -me` <*hostname*>.

If you don't intend to manage `mem_free` as a consumable resource, make it a non-consumable resource again in the `consumable` column of `complex(5)` by using `qconf -mc` *host*.

- **Problem** – `qrsh` won't dispatch to the same node it is on. From a `qsh` shell:

```
host2 [49]% qrsh -inherit host2 hostname
error: executing task of job 1 failed:

host2 [50]% qrsh -inherit host4 hostname
host4
```

  - **Possible cause** – `gid_range` is not sufficient. It should be defined as a range, not as a single number. The Sun Grid Engine, Enterprise Edition 5.3 system assigns each job on a host a distinct `gid`.
  - **Possible solution** – Adjust the `gid_range` using `qconf -mconf` or the `qmon` graphical user interface. The suggested range is the following.

```
gid_range                   20000-20100
```

- **Problem** – `qrsh -inherit -V` does not work when used inside a parallel job. You receive the following message.

```
cannot get connection to "qlogin_starter"
```

  - **Possible cause** – This problem occurs with nested `qrsh` calls, and is due to the `-V` switch. The first `qrsh -inherit` call will set the environment variable, `TASK_ID` (the id of the tightly integrated task within the parallel job). The second `qrsh -inherit` call will then use this environment variable for registration of its task, which will fail as it tries to start a task with the same id as the already running first task.
  - **Possible solution** – You can either unset `TASK_ID` before calling `qrsh -inherit`, or choose not to use the `-V` switch, but `-v` instead, and export only the environment variables that you really need.

- **Problem** – qrsh does not seem to work at all. You receive messages similar to the following.

```
host2$ qrsh -verbose hostname
local configuration host2 not defined - using global
configuration
waiting for interactive job to be scheduled ...
Your interactive job 88 has been successfully scheduled.
Establishing /share/gridware/utilbin/solaris64/rsh session to
host exehost ...
rcmd: socket: Permission denied
/share/gridware/utilbin/solaris64/rsh exited with exit code 1
reading exit code from shepherd ...
error: error waiting on socket for client to connect:
Interrupted system call
error: error reading return code of remote command
cleaning up after abnormal exit of
/share/gridware/utilbin/solaris64/rsh
host2$
```

  - **Possible cause** – Permissions for qrsh are not set properly.
  - **Possible solution** – Check the permissions of the following files, which are located in $SGE_ROOT/utilbin/. (Note that rlogin and rsh need to be setuid and owned by root.)

    ```
    -r-s--x--x 1 root root 28856 Sep 18 06:00 rlogin*
    -r-s--x--x 1 root root 19808 Sep 18 06:00 rsh*
    -rwxr-xr-x 1 sgeadmin adm 128160 Sep 18 06:00 rshd*
    ```

Note – The $SGE_ROOT directory also needs to be NFS-mounted with the setuid option. If it is mounted with nosuid from your submit client, then qrsh (and associated commands) will not work.

- **Problem** – When you try to start a distributed make, qmake exits with the following error message.

```
qrsh_starter: executing child process qmake failed: No such
file or directory
```

  - **Possible cause** – The Sun Grid Engine, Enterprise Edition 5.3 system will start an instance of qmake on the execution host. If the Sun Grid Engine, Enterprise Edition 5.3 environment (especially the PATH variable) is not set up in the user's shell resource file (.profile/.cshrc), this qmake call will fail.

- **Possible solution** – Use the `-v` option to export the `PATH` environment variable to the `qmake` job. A typical `qmake` call is the following.

```
qmake -v PATH -cwd -pe make 2-10 --
```

- **Problem** – When using the qmake utility, you receive the following error message.

```
waiting for interactive job to be scheduled ...timeout (4 s)
expired while waiting on socket fd 5

Your "qrsh" request could not be scheduled, try again later.
```

- **Possible cause** – The `ARCH` environment variable could be set incorrectly in the shell from which `qmake` was called.
- **Possible solution** – Set the `ARCH` variable correctly to a supported value that matches a host available in your cluster, or else specify the correct value at submit time; for example, `qmake -v ARCH=solaris64 ...`

Diagnosing Problems

The Sun Grid Engine, Enterprise Edition 5.3 system offers several reporting methods to help you to diagnose problems. The following sections outline their uses.

Pending Jobs Not Being Dispatched

Sometimes, a pending job is obviously capable of being run, but does not get dispatched. To diagnose the reason, the Sun Grid Engine, Enterprise Edition 5.3 offers a pair of utilities and options, `qstat -j` <*jobid*> and `qalter -w v` <*jobid*>.

- `qstat -j` <*jobid*>

  When enabled, `qstat -j` <*jobid*> provides the user with a list of the reasons why a certain job has not been dispatched in the last scheduling run. This monitoring can be enabled or disabled, as it can cause undesired communication overhead

between the schedd daemon and qmaster (see under schedd_job_info in sched_conf(5)). The following is a sample output for a job with the id, 242059.

```
% qstat -j 242059
scheduling info: queue "fangorn.q" dropped because it is temporarily not available
                 queue "lolek.q" dropped because it is temporarily not available
                 queue "balrog.q" dropped because it is temporarily not available
                 queue "saruman.q" dropped because it is full
                 cannot run in queue "bilbur.q" because it is not contained in its hard
                 queue list (-q)
                 cannot run in queue "dwain.q" because it is not contained in its hard
                 queue list (-q)
                 has no permission for host "ori"
```

This information is generated directly by the schedd daemon and takes the current utilization of the cluster into account. Sometimes this is not exactly what you are interested in; for example, if all queue slots are already occupied by jobs of other users, no detailed message is generated for the job you are interested in.

- qalter -w v *<jobid>*

This command lists the reasons why a job is not dispatchable in principle. For this purpose, a dry scheduling run is performed. What is special about this dry scheduling run is that all consumable resources (also slots) are considered to be fully available for this job. Similarly, all load values are ignored because they are varying.

Job or Queue Reported in Error State E

Job or queue errors are indicated by an uppercase E in the qstat output. A job enters the error state when the Sun Grid Engine, Enterprise Edition 5.3 system tries to execute a job in a queue, but fails for a reason that is specific to the job. A queue enters the error state when the Sun Grid Engine, Enterprise Edition 5.3 system tries to execute a job in a queue, but fails for a reason that is specific to the queue.

The Sun Grid Engine, Enterprise Edition 5.3 system offers a set of possiblities for users and administrators to gather diagnosis information in case of job execution errors. Since both the queue and the job error state result from a failed job execution, the diagnosis possibilities are applicable to both types of error states.

- User abort mail

If jobs are submitted with the `submit` option, `-m a`, abort mail is sent to the address specified with the `-M` *user*[*@host*] option. The abort mail contains diagnosis information about job errors and is the recommended source of information for users.

- `qacct` accounting

  If no abort mail is available, the user can run the `qacct -j` command to get information about the job error from the Sun Grid Engine, Enterprise Edition 5.3 system's job accounting function.

- Administrator abort mail

  An administrator can order administrator mails about job execution problems by specifying an appropriate email address (see under `administrator_mail` in `sge_conf(5)`). Administrator mail contains more detailed diagnosis information than user abort mail, and is the recommended method in case of frequent job execution errors.

- Messages files

  If no administrator mail is available the, `qmaster messages` file should be first investigated. You can find loggings related to a certain job by searching for the appropriate job ID. In the default installation, the `qmaster messages` file is `$SGE_ROOT/default/spool/qmaster/messages`.

  You can sometimes find additional information in the messages of the `execd` daemon from which the job was started. Use `qacct -j` <*jobid*> to find out the host from which the job was started, and search in `$SGE_ROOT/default/spool/`<*host*>`/messages` for the jobid.