Sun™ Grid Engine, Enterprise Edition—Software Configuration Guidelines and Use Cases

Charu Chaubal, Grid Computing

Sun BluePrints™ OnLine—July 2003

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 817-3179-10
Revision A 6/18/03
Edition: July 2003

Please

Sun™ ONE Grid Engine, Enterprise Edition—Software Configuration Guidelines and Use Cases

This article describes a set of use cases for configuration of Sun™ Grid Engine, Enterprise Edition 5.3 (Sun ONE GEEE) software. It is meant to be a starting point from which intermediate to advanced Sun One GEEE software administrators can create a customized configuration for their particular environment. It is important to realize that each environment has unique requirements.

The greatest benefits of the Sun ONE GEEE software policy module are obtained by fine-tuning a configuration once the results of the initial configuration have been assessed. Moreover, as the environment evolves and the needs of the enterprise change, additional tuning on an ongoing basis will probably be appropriate.

This article assumes the reader has some familiarity with the Sun ONE GEEE features and parameters. For the details of the policies and the major parameters used to set up and influence the various policies, consult the *Sun ONE Grid Engine, Enterprise Edition 5.3 Administration and User's Guide*, Part IV, Chapter 9. For a complete list of parameters, consult the *Sun ONE Grid Engine 5.3 and Sun ONE Grid Engine, Enterprise Edition 5.3 Reference Manual*.

Policy Use Cases

This section covers situations in which individual Sun ONE GEEE software policies are implemented to achieve certain goals. The focus is on cases in which different policies are combined.

Job Types With Groups

In this scenario, there are two or more enterprise-wide groups (for example, departments, projects, user groups, and so forth). Each group must receive a specified share of the resources, averaged over time. In addition, there are two or more priority types of jobs that must be dispatched in strict priority order. High-priority jobs are always scheduled ahead of medium-priority jobs, which are scheduled ahead of low-priority jobs regardless of the group from which they originate.

For simplicity, this article describes the solutions with only two groups and two job-types. and assumes that the desired resource allocation ratio between the groups is 75:25, and that the job types can be categorized as either normal or high, indicating relative priority. It should be straightforward to extend the concept to multiple groups and job types.

Two approaches can be used to implement this scenario.

Approach Number 1—Projects With Overloading

In this approach, every combination of {`group, job-type`} is assigned a unique project. For example, with two groups, Group A and Group B, and two types, normal and high, the projects would be:

- grpAhigh
- grpAnrml
- grpBhigh
- grpBnrml

When submitting a job, the user specifies which combination {`group, job-type`} they want to assign to it by using the `-P` option to `qsub`. Hence, the concept of *project* is *overloaded* with two characteristics: group and job type. Users would need to be trained to use the appropriate option to the submit command, for example, a user in Group A would run a high-priority job as follows:

```
qsub -P grpAhigh myjob.sh
```

To prevent users from accidentally (or deliberately) specifying the wrong group, the access lists for the projects can be set to explicitly include or exclude certain users or user groups. See man page `access_list`(5).

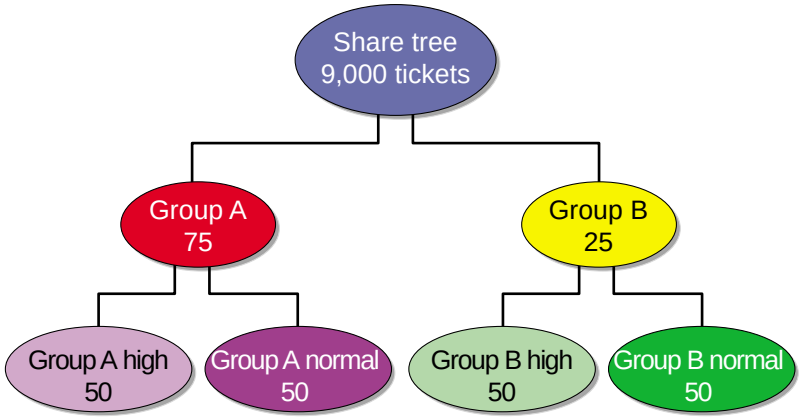

FIGURE 1 Share Tree Policy Share Assignments for Approach Number 1

After creating SGEEE software projects as described previously, you must create a share tree as shown in FIGURE 1. See man page `share_tree`(5). The penultimate nodes in the tree correspond to the enterprise-wide groups and the leaves are the different projects grouped accordingly. The shares for each node group are set to match with the desired allocation ratio. The shares assigned to the project leaves can all be set the same, because you are not interested in tracking the difference in *cumulative* utilization between high- and medium-priority jobs. Rather, you want to ensure that the higher priority jobs are *always* given greater precedence.

The next step is to add override tickets to the projects according to their priority. In this example, the following assignments would be made:

TABLE 1 Assignment of Override Tickets to Projects

Project Name	No. of Override Tickets
grpAhigh	10,000
grpBhigh	10,000
grpAnrml	0
grpBnrml	0

Since there are only two job types, it is sufficient to give a certain number to the highest- priority job, and zero to the lowest. If there were more job types, they would be allocated tickets such that the difference between the levels is always 10,000, for example 0, 10000, 20000. Essentially, a *priority band* is set for each distinct priority; the number of tickets give the ranking of the bands. The number 10,000 has a significance that is explained in the following paragraphs.

In conjunction with setting the override tickets for each project, the scheduler parameter SHARE_OVERRIDE_TICKETS must be set to FALSE under schedd_params. See man page sge_conf(5). This setting ensures that the tickets do not get divided among the jobs of each project, but rather, each job will get the full 10,000 project override tickets that are necessary to implement the priority bands.

The final step is to assign 9,000 tickets to the share tree policy. The reasoning behind this is as follows. The share tree policy allocates tickets to jobs according to the cumulative utilization of the individual projects, as compared with their share assignments. In the extreme case, in which one project's cumulative utilization is almost zero (and the compensation factor is set to one), a single job submitted into that project could get allocated all 9,000 tickets. Nevertheless, if another job from a high-priority job type is submitted, the 10,000 override tickets will be sufficient to *override* the 9,000 share policy tickets, and because it has more tickets overall, it would go ahead in the pending list.

More generally, the number of share policy tickets should always be less than the "difference" between the numbers of tickets assigned to the levels of priority for the job types. This is why 10,000 tickets was chosen as the difference between job type levels, while 9,000 was chosen as the total allotment of the share policy.

Note that the actual numbers do not have any significance to the Sun Grid Engine scheduler. The figures 9,000 and 10,000 are simply easy to understand and manage.

Approach Number 2—Projects Map Job Types

An alternative way to configure this scenario is to use projects to map only the job types, and put all of the group information into the share tree. The first step in this approach is to create a project for every job priority type. In this example, we would have two projects, with the number of override tickets again configured to give priority bands. As with the previous example, set the scheduler parameter SHARE_OVERRIDE SHARES to FALSE.

TABLE 2 Project and Override Ticket Assignments

Project Name	No. of Override Tickets
normal	0
high	10,000

The next step is to create a share tree with the desired groups and share allocation, as shown in FIGURE 2.

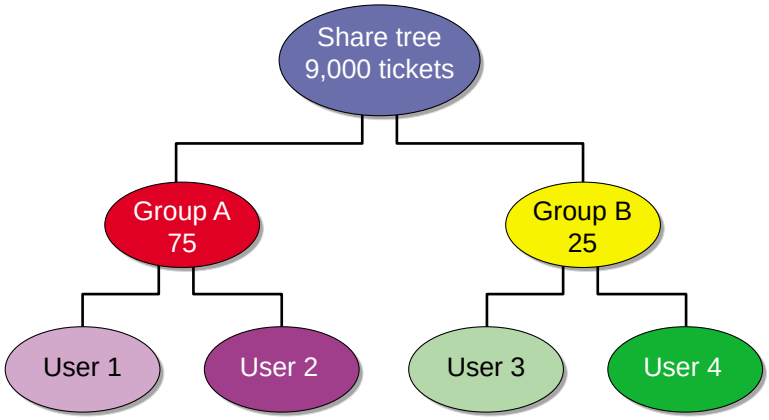

FIGURE 2 Share Tree Policy Share Assignments For Approach Number 2

▼ To Add Users

The critical part, and the one that requires most attention, is to explicitly add every user to the appropriate location in the tree. This is a two-step procedure:

1. **Create a Sun ONE Grid Engine user object for every user. See man page** `user`(5)**.**
2. **Assign the Sun ONE Grid Engine user object to the proper place in the tree.**

As before, a total of 9,000 tickets are assigned to the share tree policy; in other words, a number smaller than the difference between the number of override tickets for the different priority levels.

When submitting a job, users only need to specify the job type by using the `-P` option to `qsub`. Users do *not* need to be trained to specify a group, for example, a user in Group A would run a high-priority job as follows:

```
qsub -P high myjob.sh
```

Since the user was explicitly placed in the share tree under a particular group, the utilization by jobs from that user are automatically accounted correctly.

Comparison of Approaches

The advantage of approach number two is that it is simpler for users, since they only need to specify the job type when submitting jobs without worrying about specifying the proper group. The disadvantage is that it is more work for the administrators to set up, because they must explicitly add every user to the share tree. For an environment with a large number of users, this is best achieved via scripting, and integrating with some external user list. For example, there might be a Lightweight Directory Access Protocol (LDAP) directory that contains users organized into departments. You could write a script that reads in user's information from this directory, creates the Sun ONE GEEE user object, and then inserts the user object into the tree depending upon the department code. An example listing of such a code is given in CODE EXAMPLE 1. Such a procedure would need to be done any time a user is added to the environment.

Projects Span Groups

In this scenario, there are two or more enterprise-wide projects, that is, sets of jobs that are closely related, and two or more groups of people with different privileges, working on both projects together. These groups could be, for example, from different departments, or there could be *regular* and *power* users, the latter having greater privileges. The desire is to allocate resources to the projects based on cumulative utilization, while simultaneously guaranteeing a certain priority or service level for the different groups. For example, power users' jobs go before other users or else they receive a greater proportion of available resources.

Configuration

This example assumes two projects, Project 1 and Project 2, and two groups, Department A and Department B. See man page `access_list`(5). The goal is to give 20 percent of resources to Project 1 and 80 percent to Project 2. In addition, people in Department A should get 60 percent of the resources, regardless of which project they submit a job under, while Department B should get 40 percent. (Later, we will modify this for the case where Department A's jobs should always go ahead of Department B's jobs).

▼ To Set Up the Share Tree Policy

The procedure is to create the two projects and then set up the share tree with the desired resource allocation ratio among the projects.

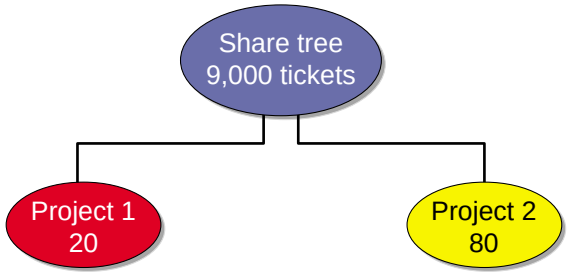

FIGURE 3 Share Tree Policy Share Assignments for Projects That Span Groups

1. **Set up two Sun ONE Grid Engine departments (**TABLE 3**) in the Userset/Userlist configuration.**

TABLE 3 Department Setup and Share Assignment

Department Name	No. of Functional Shares
Dept A	60
Dept B	40

2. **In the functional policy configuration, assign shares to the two departments accordingly.**
3. **Set the number of tickets for the share tree policy to 1,000,000 and the number of tickets for the functional policy to 1,000**
4. **Set the following parameters in the cluster configuration under** `schedd_params`:

```
POLICY_HIERARCHY=FS,SHARE_FUNCTIONAL_SHARES=TRUE
```

To submit jobs the users would simply specify the project under which the job's utilization should be accounted:

```
qsub -P Project1 myjob.sh
```

You can restrict access to projects using project access lists.

This example shows how two policies can be combined to achieve a desired goal, and also illustrates one use of the POLICY_HIERARCHY parameter. With this setup, jobs are balanced between projects according to the specified resource allocation ratio but, *within* a project, jobs are dispatched according to functional (that is,

department) ordering. Because the parameter SHARE_FUNCTIONAL_SHARES is set to TRUE, it prevents one department from excluding the other. Instead, jobs are dispatched in an interleaved fashion among all of the departments.

Most crucially, the share policy is guaranteed to have highest precedence by two factors:

- The number of share policy tickets greatly exceeds the number of functional policy tickets
- The POLICY_HIERARCHY is set to FS.

Having more share policy tickets than functional tickets means that the tickets allocated by the share policy will have the greatest impact on the overall number of tickets assigned to each job, which determines the final dispatch order. The number of tickets from the functional policy assigned to each job will be so small that, in most cases, it will be negligible in determining the total number of tickets assigned to each job.

The functional tickets only have an overriding influence in the extreme case where there is a very large number of pending jobs, or when the utilization of a project greatly exceeds the target. In this case, the number of share policy tickets assigned to a particular job might be very low, lower perhaps than the number of functional policy tickets assigned to the same job. Additionally, when a large mismatch between actual and target utilization for a project exits, the compensation factor can be used to limit the degree to which a project's ticket allocation is diminished to distribute the share policy tickets more evenly between projects.

Nevertheless, we want to ensure that, even though the number of functional tickets is small compared with the share policy, the functional policy should still have some impact. This is where the POLICY_HIERARCHY setting comes in. Setting FS instructs the scheduler to consider the functional tickets *first*, for sorting *within* a share tree node. Thus, instead of first-in first-out (FIFO) ordering, jobs *within* a share tree node are ordered according to the functional policy settings.

Configuration Guidelines

The scenarios in the previous section gave examples of how the various parameters in the Sun ONE GEE are used to effectuate a desired resource allocation scheme. This section summarizes a few principles to apply when customizing the policies for an arbitrary scenario.

Policy-Setting

This section contains guidelines for:

- Number of Tickets
- Ticket Sharing Settings
- Share Tree Policy

Number of Tickets

The relative number of tickets determines which policy is overarching versus fine tuning. To make one policy clearly dominate over the other, ensure that the difference in number of tickets between the two polices is large. If not, then the two policies could contribute roughly the same number of tickets to jobs, and the final outcome would be hard to predict. This is particularly relevant when using the `POLICY_HIERARCHY` to specify the precedence of policies. The policy that comes earlier in the hierarchy should have fewer tickets than the policy that comes later.

Ticket Sharing Settings

Three `SHARE_*_*` parameters in `schedd_params` influence the overall behavior of the policies:

- `SHARE_FUNCTIONAL_SHARES` determines if you want strict or interleaved ordering. If set to `FALSE`, the net effect is that whatever share values are set in the functional policy will be interpreted as strictly an ordering. A setting of `TRUE` causes the share values to be determined as an allocation ratio, with jobs dispatched in an interleaved fashion to result in the specified ratio.
- `SHARE_OVERRIDE_TICKETS` and `SHARE_DEADLINE_TICKETS` prevent these respective policies from “taking over” the scheduling system. If these are set to FALSE, the number of override or deadline tickets in the system increases with the number of jobs submitted. If these jobs are important, this is the desired behavior. However, if these parameters are set to TRUE, the number of tickets is fixed, and submitting more jobs dilutes the number of tickets per job assigned by these two policies. This can, for example, help prevent “abuse” by users who are granted deadline or override privileges. The choice depends on the amount of authority the users should have.

Share Tree Policy

If the share-tree policy is to be used *and* you wish to allocate and track usage on a per-user basis, *every user must have a user object in the Sun ONE GEEE software*. The reason is that the Sun ONE GEEE software must create a data structure in which to

track and store the resource allocation usage for each user. The exception to this is if you set up the *default user* under each project. This action lets you set the resource allocation for *generic* users; you then only need to add users whose resource allocation differs from the default.

If, however, you want to allocate and track usage on a per-project basis only, there is no need to add every user as a Sun ONE GEEE user object. Users simply submit jobs using the
`-P` *project* flag. Privileged projects can be restricted using project access lists.

To simplify the management of users within Sun ONE GEEE software, the configuration command `qconf` has a rich set of options that allow every operation in Sun ONE GEEE software to be scripted. The following script (CODE EXAMPLE 1) is an example of how you might use scripting to populate a share tree based upon an LDAP directory. For complete list of possibilities, consult the man page and `-help` option of `qconf`.

CODE EXAMPLE 1 Sample Share Tree Updating Automation Script

```
#!/bin/ksh
# example script to add users to an already-existing
# SGEEE share tree based on the enterprise's
# LDAP directory entries
# nf: a command which displays information from
# an LDAP directory
# NOTE: use an equivalent command for your site
usage () {
         echo "Usage: $0 <dept_code> <sharetree_nodename>"
}
add_sgeee_user() {
          TMP=/tmp/sgeee.$$
          sgeuser=$1
          echo "name $sgeuser" > $TMP
          echo "oticket 0" >> $TMP
          echo "fshare 0" >> $TMP
          echo "default_project NONE" >> $TMP
          qconf -Auser $TMP
          rm $TMP
}
if [ $# -ne 2 ] ; then
          usage;
          exit 1;
fi
DEPT=$1
NODE=$2
# below is a command which extract usernames from
# the LDAP directory based upon department codes
# NOTE: strip the line which simply tells
```

CODE EXAMPLE 1 Sample Share Tree Updating Automation Script *(Continued)*

```
#!/bin/ksh
# the number of entries found
USERS=`nf -D $DEPT -c u | grep -v "entries found"`
for user in $USERS; do
            add_sgeee_user $user
            qconf -astnode /$NODE/$user=50
done
```

Prototyping a Scenario

Configuring Sun ONE GEEE software is an *iterative* process. By this we mean that you should not try to achieve a given final result immediately (unless it is relatively simple). Instead, the approach should be to implement a trial configuration and, after testing it, refine it further and repeat this procedure as needed. The best way to start this iterative process is to create a prototype of your actual environment. This process would involve measures such as:

- Dedicating a small number of systems for the prototype (three or four systems are sufficient)
- Creating dummy jobs that emulate how the actual production jobs would behave (unless you can actually use your production applications for the prototyping)
- Creating dummy users, projects, departments, and so forth

After configuring a Sun ONE GEEE setup candidate, a quick way to see if it is behaving as expected is to suspend or disable all queues, and then submit jobs according to the expected usage pattern. Using the `qstat -ext` command, you can inspect the number of tickets assigned to each job and the contribution to the total that is coming from the individual policies. Since the overall total number of tickets determines the final job dispatch order, you can see, for example, if a certain policy is contributing too many or too few tickets to this total, and readjust the policy parameters accordingly.

Other Configuration Policies

You should keep in mind the fact that the Sun ONE GEEE software has other capabilities beyond the policy module which can help to create the configuration that suits a given scenario.

- User lists and departments can be used to control access rights to queues, hosts, and projects; for example, permit only certain jobs or users to utilize certain systems.

- Preemption using subordinate queues can provide the ability to run jobs that have *immediate* priority; for example, very important jobs or interactive jobs.
- Calendars for `suspend` or `disable` can be used to disable some queues selectively, while leaving others enabled; for example, low-priority projects can run at night, while higher priority projects can run any time.
- Queue sort method (load formula or sequence number) can be used to sort among eligible queues to determine the order in which resources (queues) are selected for jobs.
- Administrator-defined complexes and resources provide the ability to manage jobs based upon practically any characteristic or metric. Load sensors complete the picture by providing a way to input the current value of a metric into the system.

These features are present in the basic Sun ONE Grid Engine software, but taken together with the policies of Sun ONE Grid Engine, Enterprise Edition, the possibilities for adapting the software to suit a given environment's needs are indeed great.

Extended Use Case

The use case described in this section shows how you can combine the Sun ONE GEEE allocation policies with other management features to satisfy a more complicated scenario.

The following is an overview of the scenario:

- Two kinds of jobs, ranked by priority
- A large number of single-CPU systems, ranked by CPU speed, for example, *host A*: 900MHz, *host B*: 750MHz, *host C*: 464MHz,..., *host H*: 250MHz
- High-priority jobs are to go to the fastest system and low-priority jobs go to the slowest system.
- High-priority jobs should have greater percentage of resources allocated to them. However, a job should always run if a system is available, even if it means that a high- priority job goes to a slow system because all fast systems are occupied.
- Multiple teams (departments), and each team has a different share of the two job types; for example, team 1 equals 75 percent high, 50 percent low, team 2 equals 25 percent high, 50 percent low, and so forth.

Sample Outcome

If a user submits five high-priority jobs and one low-priority job, and host E is already occupied, the following dispatch order is seen:

TABLE 4 Dispatch Order

Job	Host
high:job 1	host A
high:job 2	host B
high:job 3	host C
high:job 4	host D
occupied	host E
high:job5	host F
low:job 1 host H	

Configuration

For simplicity, assume that:

- Four Sun ONE GEEE projects (*high_1, high_2, low_1, low_2)* are set up.
- Two Sun ONE GEEE departments (*Dept1, Dept2)* are set up.

▼ To Configure the Extended Use Case

1. **Set up the share tree configuration as shown in** FIGURE 4**.**

   The distinction between high and low is not made here, but in the override policy in the next step.

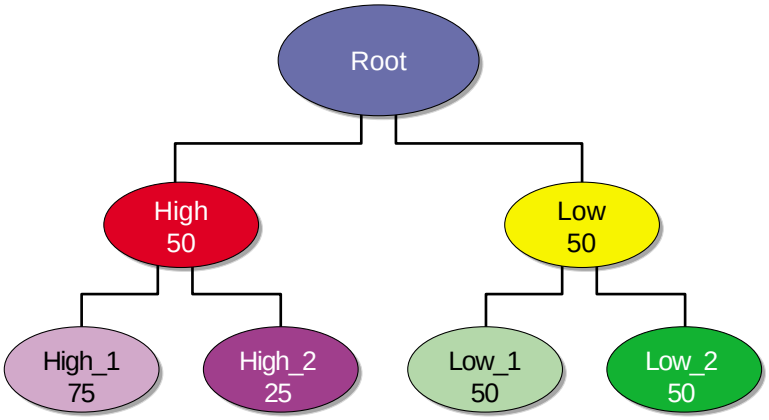

FIGURE 4 Share Tree Policy Assignments for Extended Use Case

2. **In the override policy, set the following:**

TABLE 5 Override Policy

Project	Override Tickets
high_1	200,000
high_2	200,000
low_1	100,000
low_2 100,000	

3. **In the cluster configuration, set the following for** `schedd_params`:
   - `SHARE_OVERRIDE_TICKETS=FALSE` (This will cause **all** *high* jobs to go ahead of *low* jobs) or `SHARE_OVERRIDE_TICKETS=TRUE` (This will cause *high* and *low* jobs to dispatch in a 2:1 ratio.)
   - `POLICY_HIERARCHY=SO`
4. **Define two queues per host, each with a single slot; for example,** *hostname.x* **and** *hostname.y***.**

5. **Set the queue number as follows:**

TABLE 6 Queue Number Setup

Hostname	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
hostname.x **queue No.**	1	2	3	4	5	6	7	
hostname.y **queue No.**	8	7	6	5	4	3	2	1

6. **For every host, set the host-level** `slots` **parameter to 1 to ensure that only one slot in total is ever occupied on a single host. See man page** `complex`(5)**.**
7. **Grant the access rights as follows:**
   - *hostname.x* queues are only accessible to high_1, high_2.
   - *hostname.y* queues are only accessible to low_1, low_2.
8. **Set scheduler configuration to "sort by queue number" See man page** `sched_conf`(5).
9. **Grant access for the departments in the Project Configuration as follows:**
   - Dept1 can submit to high_1 and low_1
   - Dept2 can submit to high_2 and low_2

User Instructions

▼ To Issue User Instructions

- **Submit job using the** `-P` **flag to indicate the priority.**

  For example, members of Dept 1 can do: `qsub -P high_1 myjob.sh` for high priority jobs and `qsub -P low_1 myjob.sh` for low priority jobs.