

SCHEDULER POLICIES FOR JOB PRIORITIZATION IN THE SUN N1™ GRID ENGINE 6 SYSTEM

Charu Chaubal, N1 Systems

Sun BluePrints™ OnLine — October 2005



© 2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054 USA

All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California.

Sun, Sun Microsystems, the Sun logo, Sun BluePrints, N1, Solaris, and SunDocs are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a). DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS HELD TO BE LEGALLY INVALID.



Please
Recycle



Adobe PostScript

Table of Contents

Scheduler Policies for Job Prioritization in the Sun N1™ Grid Engine 6 System	1
Distributed Resource Management Scheduling	2
Implementation of Scheduling Schemes	3
Scheduling Policies	4
Entitlement (also known as the Ticket Policy)	4
Urgency	5
Custom	6
Combining Policies	6
Observing Scheduling Policies in Action	7
Scheduling and Queues	7
Use Case 1: Fair Share with Prioritization	8
Variations	11
Use Case 2: Automatic Priorities based on Job Requirements	12
Use Case 3: Prioritization with Preemption	14
Variations	17
Summary	18
About the Author	18
Acknowledgements	18
References	18
Ordering Sun Documents	19
Accessing Sun Documentation Online	19

Scheduler Policies for Job Prioritization in the Sun N1™ Grid Engine 6 System

Grid engine technology powers collections of network-connected servers, called grids, providing efficient use of computing resources. The Sun N1™ Grid Engine 6 software, the newest version of Sun's resource management solution, includes the core services for establishing and managing a grid environment, and provides policy-based workload management and dynamic provisioning of application workloads for increased productivity. This article describes the tools and techniques for resource management that are available in the Sun N1 Grid Engine 6 software, and explains how to use them effectively. It discusses the prioritization policies in the Sun N1 Grid Engine 6 software, describes how they fit with the new resource aggregation methods, and makes recommendations for how to map real-life resource allocation schemes to N1 Grid configurations.

The article addresses the following topics:

- “Distributed Resource Management Scheduling” on page 2 describes how the Sun N1 Grid Engine 6 software implements job scheduling.
- “Scheduling Policies” on page 4 describes the various scheduling policies that can be employed in an N1 Grid.
- “Use Case 1: Fair Share with Prioritization” on page 8 details an example scenario providing fair share use of resources with prioritization of jobs.
- “Use Case 2: Automatic Priorities based on Job Requirements” on page 12 illustrates automatically determining priorities based on job requirements.
- “Use Case 3: Prioritization with Preemption” on page 14 explains how to provide prioritization of jobs in combination with preemption of lower priority jobs.

This article assumes that the reader has a basic understanding of the N1 Grid system and the services it provides. A previous Sun BluePrints™ article, *Using Host Groups and Cluster Queues in the Sun N1 Grid Engine 6 System* (August 2005), discusses abstracting collections of resources within the N1 Grid architecture using cluster queues and host groups, and explains how these features can be used to simplify administration and implement scheduling policies. In addition, please see the *N1 Grid Engine 6 Administration Guide* and the *N1 Grid Engine 6 User's Guide* for more detailed information on using the Sun N1 Grid Engine 6 software.

Distributed Resource Management Scheduling

The *scheduler* is a fundamental component in a distributed resource management system that is responsible for several coordinated decisions: It decides which jobs to run, and where and when to run them (see Figure 1). This process usually takes place in an endless loop, with each pass through the loop often referred to as a *scheduler run*.

The scheduler in the N1 Grid system receives the following information as input:

- *A list of all the jobs currently in the Grid*

This list includes all active (running or suspended) and pending jobs. Included with each job are data about the job, such as: resources requested by the job (including hard requests, soft requests, plus any default resource requests), identity of the job submitter (user, project, department, etc.), and specification of the job's execution context (in Sun N1 Grid Engine 6 software terms, the queue and parallel and/or checkpoint environment information).

- *Current status of hosts¹ in the Grid*

For the compute hosts (hosts which execute jobs), the current state is described by the values of all resources being tracked by the N1 Grid software. This information includes both built-in resource tracking (e.g., load average and memory usage) and custom resource tracking (e.g., network activity and free disk space). The scheduler also receives information on the available slots, indicating the maximum number of jobs that can be run simultaneously. This capacity is affected by the number of jobs currently active on the host as well as the status of load and suspend thresholds.

- *Current status of globally-available resources in the Grid*

Global resources that are often tracked by the N1 Grid software in real customer environments include floating software licenses and available space on a shared storage partition.

- *Past grid usage*

The scheduler can often rely on state information, taking into account the past history of what has transpired on the Grid to influence or direct its future decisions.

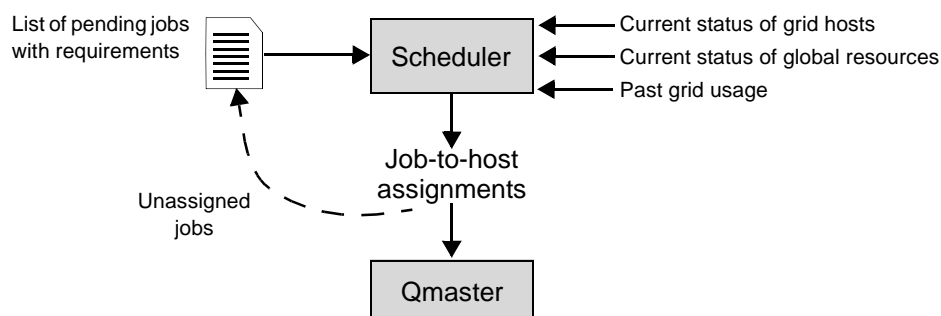


Figure 1. Job scheduling in the N1 Grid system.

1. Technically, the Sun N1 Grid Engine 6 software looks not at compute hosts, but at queue instances (the instantiation of a queue on a compute host).

Given this information, the scheduler then makes a series of decisions:

- First, it filters jobs that can be started (those that are not in a hold state, do not depend on other jobs, and have no start-time assigned).
- Second, it computes the job priority for all available jobs.
- Next, it decides what jobs are actually possible to run at the present time based on the hardware requirements of the job and the availability of any host to fulfill those requirements.
- Finally, the scheduler uses this priority ordering to select jobs and determine where to send those jobs, based upon the global scheduling preference as well as any job-specific soft requests.

This selection process continues until all available job slots have been claimed by a job. The output from the scheduler is the final set of job dispatch orders, which get sent to the Qmaster component in the N1 Grid system for actual enforcement. Those jobs that did not get assigned to a compute host wait until the next scheduler run decides their fate.

Jobs that specify a resource reservation are treated specially by the scheduler in the N1 Grid system. For example, a job may require a software license or may need to run across 128 compute hosts simultaneously. For jobs that specify required resources, the scheduler sets aside the necessary resources for them at a particular point in time. When that point arrives, the reserved resources are given to the job.

Implementation of Scheduling Schemes

One focus of many schedulers is to influence the *dispatch priority* of the jobs, the order in which pending jobs are sent to compute hosts for execution. The dispatch priority can be used to implement a wide range of business-based policies in environments running the Sun N1 Grid Engine 6 software:

- *Precedence based on criteria such as user, job, or project*

Precedence of jobs based on criteria such as user or group hierarchy, job category, or project is done by always assigning the same fixed dispatch priority based on the specific policy. For example, jobs from one department can always have a higher dispatch priority than those from another department.

- *Sharing of compute resources*

Sharing of the compute resources on the grid can also be achieved by modifying the dispatch priority. Some usage metric, typically CPU seconds, is chosen for determining the sharing policy. A sharing scheme is also specified with a share-tree structure, whose nodes represent groups of users or projects, along with weights assigned to each node to indicate the desired sharing. The dispatch priority is then modified to target the specified sharing policy (even if the desired sharing is not exactly achieved). Jobs associated with nodes that are below target are given a higher dispatch priority than those from nodes that are above target. This priority is continuously re-evaluated as usage data accumulates.

- *Importance of jobs determined by the resources they use*

The dispatch priority can be used to automatically determine the importance of a job by the resources it uses. For example, if a job requests the use of an expensive software license, it can automatically be given higher priority than other jobs, so that the license use is optimized.

The Sun N1 Grid Engine 6 software can implement these sophisticated business policies because it employs a centralized scheduler. Capabilities such as optimal job placement, policies for global prioritization and sharing, satisfaction of service level objectives, and management of global resources are only possible if they are managed via some single point of coordination. An alternative to the centralized scheduler approach to job and task management is to have compute hosts pull jobs from a central job repository without mutual coordination. This approach can have certain advantages, mostly in terms of performance. However, unless this alternate approach is enhanced by some type of explicit central coordination scheme, it lacks the ability to make the same kind of sophisticated scheduling decisions afforded by the centralized scheduler in the Sun N1 Grid Engine 6 software.

Scheduling Policies

The Sun N1 Grid Engine 6 software employs three top-level policies — entitlement, urgency, and custom policies — that can be configured to implement a range of scheduling concepts. Contributions from these three top-level policies are combined to assign a final dispatch priority to a job. In the N1 Grid system, the dispatch priority is simply a decimal number, with higher values denoting higher priority.

Once configured, these policies can generally control scheduling in N1 Grid systems without requiring further administrator intervention. However, fine-tuning of the configuration may be required occasionally to help ensure that the chosen policies continue to best meet the user and site requirements.

These three scheduling policies, and the approach for combining these policies to derive a final job priority, are described in the following sections. For additional information on these scheduling policies, please refer to the *N1 Grid Engine 6 Administration Guide*.

Entitlement (also known as the Ticket Policy)

The entitlement scheduling policy supports the implementation of business decisions based upon the owner of the job. The job owner can be identified as an individual user, a user group, a department, or a project. The entitlement for the owner can be a strict precedence of job dispatching, or a sharing policy can be employed.

For example, an entitlement policy can specify that jobs belonging to Project A should always run ahead of jobs belonging to Project B, regardless of past history or currently active jobs. As long as even one job from Project A is pending, it will be dispatched ahead of all pending Project B jobs (provided, of course, that there are free resources which can fulfill that job's requirements). Another entitlement policy could specify that two groups of users should share the CPU time on a grid in a 60:40 ratio. In this case, the grid software tracks the total CPU time recently used by the two groups (where the definition of recent is configurable), and the scheduler then dispatches jobs in an order that targets a 60:40 ratio of those numbers.

Entitlements in N1 Grid systems are handled using a system of *tickets*. In fact, this entitlement policy is referred to in the documentation as the *Ticket policy*.¹ The total number of tickets assigned to each job is an integer value. This value is normalized into a decimal number between zero and one, with the top of the scale determined by the highest number of tickets assigned to any job and the bottom always being zero. This normalized value becomes the entitlement contribution to the dispatch priority.

Urgency

Unlike the entitlement policy, the urgency policy accounts for only the job's individual characteristics, not its owner. The urgency value is derived from the sum of three contributions: the deadline contribution, the wait-time contribution, and the resource requirement contribution.

- *Deadline contribution*

The first component of the urgency policy, the deadline contribution, determines the influence that a job's deadline has on the dispatch priority. In the N1 Grid system, privileged users may submit their jobs with a future date and time given as the deadline. The scheduler then uses the time left until the deadline in the following formula to determine the deadline urgency (U_{deadline}):

$$U_{\text{deadline}} = \frac{W_{\text{deadline}}}{T_{\text{deadline}} - T_{\text{current}}}$$

where both the deadline time (T_{deadline}) and the current time (T_{current}) are given in Unix time (in seconds), and W_{deadline} is the deadline weighting factor. Thus, as the deadline approaches, the deadline urgency approaches the value of the weighting factor. When the deadline is reached, the urgency is pegged at the weighting factor value until the job is dispatched.²

- *Wait-time contribution*

The wait-time contribution to the urgency policy uses the time that the job has been pending to set the wait-time urgency, U_{wait} , using the following formula:

$$U_{\text{wait}} = T_{\text{wait}} \times W_{\text{wait}}$$

where the time spent since being submitted (T_{wait}) is given in seconds and W_{wait} is the wait-time weighting factor. This policy can be used to prevent low-priority jobs from being “forgotten.” If a job waits long enough, it will eventually get a high enough dispatch priority and run.

- *Resource requirement contribution*

The third contribution to the urgency policy determines the dispatch priority of the job based on the resources it has requested. As described in the *N1 Grid Engine 6 Administration Guide*, all built-in and

1. The ticket system of the Sun N1 Grid Engine 6 (N1GE6) software has been described extensively in documents that discuss the policy system of Sun Grid Engine Enterprise Edition (SGEEE) 5.3, the predecessor to N1GE6. In SGEEE5.3, the ticket policy was the only policy available; in N1GE6, it is now one of the three top level policies. Since the ticket policy is largely unchanged from SGEEE5.3, please refer to earlier publications such as the Sun BluePrints article *Sun Grid Engine, Enterprise Edition — Software Configuration and Use Cases* and the SUPerG paper *N1 Grid Engine 6 Features and Capabilities* that details the differences between N1GE6 and SGEEE5.3

2. The deadline policy used to be a part of the ticket system in Sun Grid Engine Enterprise Edition 5.3; this is no longer the case with the Sun N1 Grid Engine 6 software release.

custom resources defined in an N1 Grid system can have an urgency value associated with them. For each job, the urgency values for all requested resources are summed together to determine the resource urgency. In the case of a consumable resource, such as memory, the urgency value is multiplied by the amount of the request (e.g., the size of the memory requested). If the job is parallel, then this result is further multiplied by the number of requested execution slots for the job. (See the *N1 Grid Engine 6 Administration Guide* for more information on consumable and other resource types.)

All three contributions to the urgency policy — the deadline contribution, the wait-time contribution, and the resource requirement contribution — are added together for each job, resulting in a number that determines the overall urgency dispatch priority. This value is then normalized to a decimal number between zero and one, and this final value is assigned to the job.

Custom

The final top-level scheduling policy, the custom policy, is also referred to as the POSIX priority policy. This policy simply takes an arbitrary integer value between -1023 and 1024 and translates that value into a priority between zero and one. The POSIX priority can be assigned to a job when it is submitted, but ordinary users are only allowed to specify negative priority values that lower a job's priority. Privileged users can submit jobs with a positive priority, and they can also modify the priority value of pending jobs, assigning them any value in the entire range.

The POSIX priority policy provides a means for local sites to set up their own custom prioritization scheme. For example, a process could scan the list of pending jobs from time to time, and assign each job a unique POSIX priority based on some external consideration. A more common use of this policy is to allow an administrator to manually override the priorities that are automatically calculated by the other two scheduling policies.

Combining Policies

The final dispatch priority assigned to all pending jobs is determined by combining the contributions from the entitlement, urgency, and custom policies using the following formula:

$$P = N_e \times W_e + N_u \times W_u + N_c \times W_c$$

where P is the dispatch priority, N_e is the normalized entitlement priority and W_e is the entitlement weighting factor, and N_u , W_u , N_c and W_c are defined similarly for the urgency and custom priorities.

The choice of the weighting factors determines the precedence of the policies. Since each individual contribution is always normalized to a decimal number between zero and one, using weighting factors that are factors of ten apart from each other allows a clean and straightforward specification of policy precedence. For example, assume W_e is 0.1, W_u is 1, and W_c is 10. Since by default, all jobs are submitted with a POSIX priority of zero, the priority determined by the urgency policies ($W_u=1$) takes precedence in this example over the priority determined by the entitlement policy ($W_e=0.1$). For those jobs which have the same urgency, the ticket policy determines their relative priority. A privileged user could alter the POSIX priority of pending jobs to intervene manually in the scheduling scheme, boosting the POSIX priority of a job in order to assure that it has the absolute highest dispatch priority.

Table 1 lists the names of the weighting factors referenced in this text and the equivalent scheduler parameters used by the Sun N1 Grid Engine 6 software. These parameter names are used when modifying the scheduler configuration, either through the Qmon GUI or with the relevant command line commands such as 'qconf -ssconf' and 'qconf -msconf'.

Table 1. Scheduler weighting factors.

Reference in Text	Weighting Factor	Parameter Name
W_{deadline}	Deadline	weight_deadline
W_{wait}	Wait-time	weight_waiting_time
W_{e}	Entitlement (Ticket)	weight_ticket
W_{u}	Urgency	weight_urgency
W_{c}	Custom (POSIX)	weight_priority

Observing Scheduling Policies in Action

There are myriad ways in which the various scheduling policies in the Sun N1 Grid Engine 6 software can be set up to achieve desired business goals. To understand how a given set of configurations affects job dispatching, the priorities of the pending jobs can be observed. The command line tool `qstat` is designed to provide detailed information about job prioritization. Running this command without any options shows the overall final priority of each job. For each of the three sub-policies, there is a `qstat` option that provides detailed information about the calculation of that sub-policy and its contribution to the overall priority (see Table 2). By setting up different scenarios and parameter configurations, the resulting dispatch priority can be studied in detail and fine-tuned to achieve the optimal outcome.

Table 2. Qstat commands to inspect the scheduling sub policy.

Scheduling Sub-policy	Qstat Inspection Command
Entitlement	<code>qstat -ext</code>
Urgency	<code>qstat -urg</code>
Custom	<code>qstat -pri</code>

Scheduling and Queues

A queue in the N1 Grid system provides a means of defining a job's execution context. This context encompasses characteristics such as job runtime limits (e.g., memory, stack, and CPU time), control action methods (e.g., how to suspend and resume the job), and virtual job container (e.g., Solaris™ resource pools). Unlike queues in many other resource management systems, a queue in the N1 Grid system is specifically not meant to be used to prioritize jobs. Rather, the extended policy system of the Sun N1 Grid Engine software is intended to be used to define priorities.

Job preemption is a special case that requires special mention. With preemption, one type of job has the ability to suspend another type in order to take over its resources on a compute host. In this case, preemption occurs independently of the priorities set by the policy system. Instead, the two job types must

be implemented each with their own queue in the N1 Grid system. The third use case (see “Use Case 3: Prioritization with Preemption” on page 14) shows how to link job priorities with preemption.

The use cases that follow explore various scheduling schemes and illustrate the concepts of prioritization, fair share, and preemption in more detail.

Use Case 1: Fair Share with Prioritization

This first use case illustrates fair share of resources with job prioritization and has three objectives:

- Prescribe a scheme for sharing the CPU utilization of hosts in a grid among different projects. This approach is sometimes referred to as *fair-share*.
- Enable a prioritization scheme for jobs so that they can be submitted with different priorities, such as *high* or *background* priority. Jobs with higher priority should be dispatched before jobs of lower priority.
- Ensure that priority has precedence over sharing. For example, suppose one project has a number of low priority tasks pending, while another project has some high priority tasks waiting. Regardless of the sharing policy, the high-priority tasks should be run first. If both projects have high priority tasks pending, then the jobs from the two different projects should be dispatched in accordance with the sharing policy.

The following section details a concrete example to illustrate the use of fair share with prioritization. This example assumes an environment where the CPU utilization is to be shared among four projects. Three levels of priorities are assumed: high, regular, and low.

Table 3 indicates the values used for the relevant scheduler parameters and describes why these values were chosen.

Table 3. Scheduler parameters for Use Case 1.

Scheduler Parameter	Value	Explanation
<code>weight_urgency</code>	0.1	Set ten times higher than <code>weight_ticket</code> to ensure greater precedence
<code>weight_ticket</code>	0.01	Set ten times lower than <code>weight_urgency</code> to ensure lesser precedence
<code>weight_posix</code>	0	POSIX priority unused
<code>share_tickets</code>	1,000,000	Arbitrarily large number to enable sharing
<code>functional_tickets</code>	0	Functional policy unused
<code>weight_deadline</code>	0	Deadline policy unused
<code>weight_waiting</code>	0	Wait time policy unused

Four projects are created in the N1 Grid system to implement the sharing policy: `project1`, `project2`, `project3` and `project4`. Access lists can be defined for each project to specify users that are allowed to submit jobs under each project. Each user can also be given a default project, if desired. Next, a share-tree object (a hierarchical definition of the share-based policy) is configured to specify the allocation of CPU usage in the Grid environment (see Figure 2). In this example, `project1` receives a 20 percent share of

CPU resources, `project3` receives a 30 percent share, and `project2` and `project4` each receive a 25 percent share.

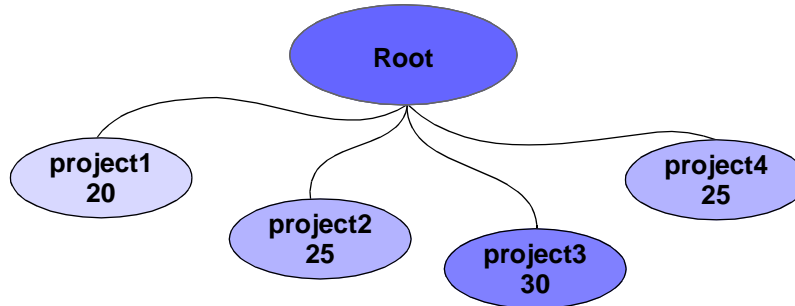


Figure 2. Share-tree structure and settings.

Note – Further details on the share-tree policy may be found in the *N1 Grid Engine 6 Administration Guide*, as well as the Sun BluePrints article *Sun Grid Engine, Enterprise Edition—Software Configuration Guidelines and Use Cases*.

Finally, the urgency policy is used to set up the priorities for this scenario. More specifically, this example use case utilizes abstract boolean resources to indicate resource-based urgency. This powerful technique allows users to tag their jobs with one of a set of predefined resource requests, and this resource is automatically translated into a priority for the job. Abstracting the details away from the user makes it simpler for them, and allows the administrator more flexibility in implementing the priorities.

The first step in configuring this scenario is to create two new custom resources, `low` and `high`, using the values listed in Table 4. These new resource definitions are added to the *grid engine system complex*, an entity that stores all pertinent information about the resource attributes that users can request for jobs. New entries can be added to the complex entity using either the Qmon graphical user interface or via the command line `qconf -mc`.

Table 4. Custom abstract resources for prioritization in Use Case 1.

Name	Shortcut	Type	Relop	Requestable	Consumable	Default	Urgency
low	lw	BOOL	==	YES	NO	0	-1000
high	hg	BOOL	==	YES	NO	0	1000

The value in the urgency column determines the urgency priority that gets assigned automatically to jobs which request the corresponding resource. These defined resources represent the low and high-priority categories defined for this environment. The third category, regular, is simply the default value for jobs which do not specify a priority. Regular jobs (i.e., jobs which don't request either resource) do not get a contribution to the urgency priority (at least not from here) and the contribution is simply zero. Recall that the final urgency priorities of the jobs are scaled to a number between zero and one, so the actual specific value for the urgency column is not important, as long as there is a monotonic difference.

Because these resources are type `BOOL`, a job can easily specify a (non-regular) priority in the submit command. Requesting a boolean resource without specifying its value automatically implies a request for a `TRUE` value. For example, the first command submits a job with a high priority, while the second command submits a low priority job:

```
# qsub -l high=TRUE jobscript.sh
# qsub -l low jobscript.sh
```

The next step is to ensure that any requests for these two new resources can be satisfied by the grid. Although these are purely abstract resources, without any physical meaning, the administrator still needs to indicate that the Grid is able to provide them. This is done by *instantiating* these resources. A resource exists only as abstract definition until it is actually given a concrete value, either on a queue, a host, or the grid as a whole. In this case, the high and low resources are instantiated on the *global* host, a special keyword in the Sun N1 Grid Engine software that indicates the resource is provided everywhere in the grid (i.e., on all hosts and all queues). This instantiation is done by modifying the special global execution host. Using either the Qmon GUI or the interactive command `qconf -me global`, the value for these two resources can be set to `TRUE` interactively. Alternatively, a single `qconf` attribute command can be used to set the value in one step:

```
# qconf -aattr execheap complex_values low=TRUE,high=TRUE global
```

This command indicates that, by default, all hosts in the grid can satisfy the request for the `low` and `high` resources.

At this point, the use case is fully implemented. Users submit jobs using the `qsub` command, specifying `high` or `low` priority, or omitting this specification for regular priority jobs. Users can also indicate the project under which their jobs should run. Alternatively, if configured by the administrator, users might belong to a default project and hence would not need to specify the project.

The urgency policy has precedence and sets the urgency contribution to the jobs' priorities to one of three normalized values, corresponding to the three possible resource urgency values (-1000, 0, 1000). Concurrently, the share-tree policy will set a ticket-based priority for every job based upon the resource sharing target. Since the urgency policy's weighting is an order of magnitude greater than the ticket policy's, it is assured that the jobs' priorities will provide the main ranking for job dispatch order. Within the groups of jobs that have the same priority, the share-tree policy will provide a secondary ordering.

Variations

A number of variations can build upon this basic approach to fair share with prioritization:

- *Relative ordering of user jobs*

Users can provide a relative ordering of their own jobs by using the '-js' (jobshare) flag to the `qsub` command. By default, the jobshare value is zero. Users simply assign a higher jobshare value to more important jobs. For example, the following command submits a job with a jobshare value of 100:

```
# qsub -js 100 myjob.sh
```

- *Administrator ability to increase job priority (directive of management)*

The POSIX policy can be used to provide a lever for administrators to arbitrarily increase the priority of certain jobs. By default, all jobs have a POSIX priority of zero. Administrators can use the `qalter` command to modify the POSIX priority value of pending jobs to a positive number, that subsequently gets translated into a large total priority. For example, modifying the `weight_posix` parameter to be one, which is ten times higher than the `weight_urgency` value, increases the priority of a given job. The POSIX priority will always take precedence over any other consideration.

- *Temporarily increasing the entitlement of a user, department, or project*

The Override component of the Ticket policy can be used to temporarily increase the entitlement of a specific user, department, or project. By assigning override tickets, the entitlement can be modified without affecting any prioritization assignments of the urgency policy. This technique essentially provides a more nuanced method for influencing the allocation of grid resources, particularly if the change is meant to reflect some business decision related to entitlement but not prioritization. For example, the override component can be used to meet the needs of specific users who temporarily need extra resource allocation because they are working on a high profile task for the next week.

- *Excluding high priority jobs from slower machines*

If the grid includes slower machines that should not run high priority jobs, it is a simple matter to exclude these jobs from those systems. Simply setting the value of the `high` resource to `FALSE` explicitly on the slower systems overrides the global `TRUE` setting. This modification can be done on each host or through the use of host groups.

- The value of the `high` resource can be set to `FALSE` on each host explicitly, either from the Qmon GUI, or the command line. A single command that explicitly lists each host can be executed by the administrator to do this:

```
# qconf -aattr execheap complex_values high=FALSE host1,host2,host3,...
```

- A host group can be used to aggregate all slower systems. Then a host group-specific value for the `high` resource can be specified in every cluster-wide queue definition. For example, the following

entry would set the `high` resource value to `FALSE` for the host group `@slowhosts` which contains all slower machines:

```
complex_values NONE,[@slowhosts=high=FALSE]
```

The first method of explicitly listing each slow host has the advantage of preventing all high-priority jobs from running on the slower systems regardless of what queues are defined now or in the future. The disadvantage is that every host must be configured explicitly, and this administrative step must be done every time a new slower system is added to the grid, thus adding administrative overhead.

The second method of using a `@slowhosts` host group has the advantage of aggregating all slower systems in one place. This host group could be re-used elsewhere for other configuration purposes. If any new slower systems are added to the grid, the only required administrative step is to add the new system to this host group. All subsequent configurations for this host would automatically be registered by virtue of being a member of this host group. The disadvantage of this method is that the host group-specific value for `high` would need to be added for every new queue defined.

In general, using host groups to organize and configure hosts is better than configuring the hosts individually. Host groups provide a powerful means of centralizing all host-based configurations in one place, enabling administrative scalability. For more on this topic, see the Sun BluePrints™ article *Using Host Groups and Cluster Queues in the Sun N1 Grid Engine 6 System*.

Use Case 2: Automatic Priorities based on Job Requirements

The importance of jobs often can be accurately determined by merely looking at the resources required by the job. Consider a typical electronic design automation (EDA) environment where most jobs require a software license from a commercial EDA application vendor. In addition, certain key tasks, such as place-and-route, require large amounts of memory and may be able to take advantage of more than one CPU. When users submit these jobs, they request the specific resources required for that job. Other tasks, such as simulation, do not have any special memory requirements and are typically single threaded. When these tasks are submitted, no special resources are needed and a software license may or may not be required.

In this scenario, jobs are prioritized using the following considerations:

- Jobs that require a more expensive software license have higher priority than jobs that request less expensive licenses, or that don't require any license. This prioritization enables the more expensive license to be as fully utilized as possible; the license should not sit unused while there are jobs pending that could make use of it.
- Jobs that require larger amounts of memory or multiple CPUs are given higher priority. These jobs tend to be critical in the overall design process flow, and any delay in running these jobs can affect many downstream jobs as well.

This prioritization scheme can be handled automatically using the urgency policies in the Sun N1 Grid Engine 6 software. Conveniently, this job-based prioritization does not require any special action or knowledge on the part of the end users. Users simply request the appropriate license consumable for their job as they normally would, along with any CPU or memory requests, and the scheduler automatically handles the prioritization decision on its own. The administrator is also free to change the urgency values of the consumables at any time without knowledge or consent from the user community, thus allowing them to fine-tune the relative urgencies of the different resources, whether they be licenses, CPUs, memory, or anything else.

Another common business policy in an EDA environment is that sharing amongst projects should have precedence over job prioritization. Often, a chip design company or division is working on several different large designs at the same time, each at different points in the project life cycle. As a result, one project might be submitting many jobs which need an expensive or demanding application, while another might not yet be at that point. Rather than giving one project more job throughput at the expense of others, these projects are simply granted a certain percentage of the CPU resources. Even if a project is not yet at the point of running an expensive or resource-intensive application, it still receives its fair share of the compute resources. Within the projects, however, the above-mentioned prioritization scheme is obeyed.

This use case is described as a variation on the first use case presented in this article (see “Use Case 1: Fair Share with Prioritization” on page 8). The projects can be set up just as in Use Case 1, with resources shared among four projects. Alternatively, the share-tree can be defined entirely in terms of a tree with intermediate nodes and user leaves that exactly specify each user’s place in the usage allocation hierarchy. For situations where users belong to only a single project, this configuration obviates the need for them to do anything special to indicate their entitlements. Table 5 indicates the values for the scheduler parameters and explains why these values are used.

Table 5. Custom abstract resources for prioritization in Use Case 2.

Scheduler Parameter	Value	Explanation
<code>weight_urgency</code>	0.01	Set ten times lower than <code>weight_ticket</code> to ensure lesser precedence
<code>weight_ticket</code>	0.1	Set ten times higher than <code>weight_urgency</code> to ensure greater precedence
<code>weight_posix</code>	0	POSIX priority unused
<code>share_tickets</code>	1000000	Arbitrarily large number to enable sharing
<code>functional_tickets</code>	0	Functional policy unused
<code>weight_deadline</code>	0	Deadline policy unused
<code>weight_waiting</code>	0	Wait time policy unused

In order to enable job prioritization based on resource requirements, the urgency value of various resources — including the built-in resources, such as memory, as well as all custom-created software license resources — should be set to a positive value. An example is shown in Table 6. The software license `license2` is more expensive than `license1`, thus it gets a higher urgency. Note that the

resource `slots` is used as a means of tracking the number of CPUs used by a job, since this ties in with the Sun N1 Grid Engine 6 Parallel Environment framework for multi-CPU jobs. For further information on using consumable resources, see the *N1 Grid Engine 6 Administration Guide*.

Table 6. Resource urgencies for prioritization in Use Case 2.

Name	Shortcut	Type	Relop	Requestable	Consumable	Default	Urgency
mem_free	mf	MEMORY	<=	YES	YES	0	100
slots	sl	INT	<=	YES	YES	0	100
license1	l1	INT	<=	YES	YES	0	200
license2	l2	INT	<=	YES	YES	0	500

For end users, submitting jobs merely involves specifying the resources required by the job. If the share-tree is defined in terms of user leaves, then the project-based entitlement is handled automatically. For an ordinary job that is not resource-demanding, it is sufficient to specify only the license needed:

```
# qsub -l license1=1 simulA1.sh
```

For a resource-intensive job, the memory and CPUs are specified in addition to the license requirement. Here, the CPU requirement of multi-threaded jobs is handled by the custom parallel environment `smp`:

```
# qsub -l license1=2 -l mem_free=2048MB -pe smp 4 placerout1.sh
```

Use Case 3: Prioritization with Preemption

Using the dispatch priority to manage jobs is not sufficient for all situations. For example, a job may need to run immediately, without waiting for a slot to free up. In this case, preemption of jobs must be configured. Jobs need the ability to force another running job to be suspended so that the processor and memory can be used exclusively by the incoming job.

A hierarchy of job priorities can still exist in this scenario, with the specification that only the highest priority jobs can preempt other jobs. Other regular priority jobs merely go ahead of the low-priority jobs in the dispatch order. Of course, the share-tree policy could remain active and work in tandem with this scheme, providing a way to arbitrate between jobs from different users with the same priority rank.

This use case describes the needs of a hypothetical facility which does weather analysis. The policy goals for this case are based upon three different types of jobs:

- *Type 1:* A meteorological application that processes satellite data once per day. This processing must occur at noon every day and is a massively-parallel job which must have exclusive access to multiple systems while it is running. In addition to the one main job, there are several pre- and post-processing jobs
- *Type 2:* Various analysis programs that run throughout the day.

- *Type 3*: A long-term climate simulation application that has only background precedence, running at night and whenever the servers are otherwise unoccupied.

The Grid utilization profile of the three types of jobs can be visualized in Figure 3.

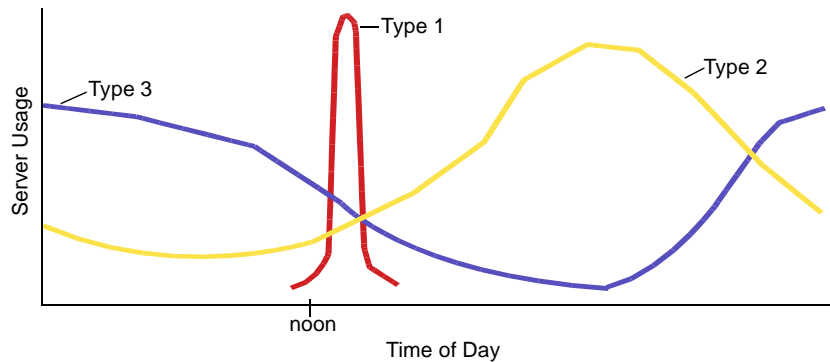


Figure 3. Server usage profile per job type.

In order to enable Type 1 jobs to run on-demand, it is necessary to create a separate queue for them. This queue preempts other jobs, that run in a *subordinate* queue. (See the *N1 Grid Engine 6 Administration Guide* for the definition and use of subordinate queues.) As alluded to earlier, preemption is one situation where the prioritization policies of the N1 Grid system are not sufficient by themselves, but must be used in conjunction with appropriate queue definitions.

The implementation of this use case requires the creation of the following objects in the N1 Grid system:

- Two queues — an immediate queue and a regular queue — created across all grid hosts, with the regular queue made a subordinate of the immediate queue.
- Two abstract boolean resources — one called `immediate` for Type 1 jobs, and one called `background` for Type 3 jobs. The settings for these two resources are given in Table 7.

Table 7. Custom abstract resources for prioritization in Use Case 2.

Name	Shortcut	Type	Relop	Requestable	Consumable	Default	Urgency
background	bg	BOOL	==	YES	NO	0	-1000
immediate	im	BOOL	==	FORCED	NO	0	1000

Since the share-tree policy is not being used in this scenario, the ticket weighting factor may be set to zero. All other scheduling parameters should be set as shown in Table 3 on page 8.

The `background` resource works the same as the `low` resource from Use Case 1: It is used to tag jobs that have background priority by assigning a negative urgency. Jobs requesting this abstract resource automatically get a lower dispatch priority versus all other jobs. Thus, these jobs will tend to run later in the day and at night, when Type 1 and Type 2 jobs have drained out of the Grid. This `background` resource should be instantiated on the `global` host, just as with the `low` resource in Use Case 1.

The `immediate` resource is used to tag Type 1 jobs that must be allowed to run immediately. Since these jobs will run only in the immediate queue, this resource is instantiated only on the immediate queue. Administrators instantiate this resource by modifying the `complex_values` attribute of the immediate queue, setting the value of `immediate` to `TRUE`. This value can be set using the Qmon GUI interface, or with a single `qconf` attribute command as follows:

```
# qconf -aattr queue complex_values immediate=TRUE immediate.q
```

This command indicates that the queue `immediate.q` can satisfy the request for `immediate`. Since this resource is not instantiated anywhere else, only the `immediate.q` queue can run these jobs.

It is important to note that for the `immediate` resource, its `Requestable` attribute has a value of `FORCED`. Any queue or host which instantiates this resource automatically has a strict requirement that only jobs which explicitly request this resource can run there. Consider the consequences if a resource with the `Requestable` attribute set to `FORCED` is instantiated on the special global host: Every single job submitted to the grid would have to explicitly request this resource, or else they would be rejected. However, in our case, we instantiate this `immediate` resource only on the immediate queue. As a result, `immediate` jobs will run only in the immediate queue, and the immediate queue will accept only `immediate` jobs.

Finally, the urgency value of the immediate resource is set to 1000 in this scenario to give it a high dispatch priority. Due to the exclusive access to the immediate queue, jobs requesting the immediate resource would never need to wait for non-immediate jobs to complete before they ran, so it would seem that setting a high urgency to force a higher dispatch priority is unnecessary. However, the scheduler in the N1 Grid system does not directly account for the subordination, or preemption, of jobs when making scheduling decisions. Nothing prevents the scheduler from dispatching immediate jobs and regular jobs (i.e., jobs which are neither immediate nor background) in the same scheduling run. As a result, an immediate job and a regular job could be scheduled on the same server, with the regular non-immediate queue suspended (preempted) right away in order to allow the immediate job to run.

A more preferable outcome would be for all regular jobs to be dispatched *after* the immediate jobs, so that they can be sent to regular queues that are not going to be suspended in the same scheduling run by immediate jobs on the same server. This result is achieved by setting the urgency value of immediate jobs to a large positive number, thus ensuring their higher rank in the dispatch order. Therefore, instead of the regular queue being suspended right after accepting a new job, it has a chance to be suspended *first* by any incoming immediate job, before it is occupied by a new non-immediate job.

Of course, it is possible that a new immediate job might arrive in the next scheduling run and be sent to a host running a non-immediate job. However, by imposing a logical and self-consistent dispatch order within a single scheduler run, the behavior of the scheduler can be made as rational as possible.

Job submission in this environment is as follows:

- To submit a Type 1 job to run at noon on August 15 (08/15):

```
# qsub -a 08151200 -l immediate job1.sh
```

The “-a 08151200” option to the `qsub` command indicates that the job should not be dispatched *before* 12:00 on 08/15. The job is dispatched in the next scheduling interval after the specified timestamp (the default schedule interval time is 15 seconds). The “-l immediate” option indicates this is an immediate job. Once the eligibility time has been reached, this job is sent to an immediate queue, preempting any non-immediate job on the same host. The job will run right away, assuming that the immediate queues are not completely occupied.

- To submit a Type 2 job:

```
# qsub job2.sh
```

- To submit a Type 3 (background) job:

```
# qsub -l background job3.sh
```

Variations

The use of the `-a` option to the `qsub` command, to dispatch a job after a specified time, is optional. Without this option, the job is dispatched as soon as it is submitted. However, for time-sensitive jobs (such as the job in the previous use case that depends on external factors) this flag is useful for placing a job into the N1 Grid system ahead of time. Using the `-a` option relieves the user of worrying about the exact job submission timing. An alternative is to submit the job with a hold, using the `qsub` command `'-h'` option, and then release the job at a later time using the `qrls` command.

Just as with Use Case 1, variations on this basic approach can be used to meet the needs of a specific N1 Grid environment. For example, users and administrators can choose to:

- Set other weighting factors appropriately to implement more fine-grained scheduling policies while still fulfilling the other more important scheduling requirements outlined in this use case.
- Use the share-tree policy to set a sharing policy.
- Use the POSIX policy to override prioritization policies for a limited set of jobs without disturbing any of the other settings.

Finally, the default preemption behavior itself can be customized. For example, instead of suspending the jobs, a checkpointing environment can be used. This approach can enable the checkpointing and restart of the jobs on another system upon preemption, or else to force the job to be killed and automatically resubmitted to start from the beginning on a different host. Restarting jobs rather than suspending them can help avoid a job being trapped in the suspended state by a long-running job.

Summary

The scheduler policies in the N1 Grid system can be configured to support a range of job prioritization decisions. Three flexible yet powerful policies — entitlement, urgency, and custom policies — work together to determine job dispatch priority. For example, these policies can be configured to set job precedence based on the submitting user or project, to share resources among multiple projects, or to automatically determine precedence based on requested job resources. When used in conjunction with cluster queues, these scheduling policies can also be used to implement job prioritization with preemption, allowing high-priority jobs to suspend lower-priority jobs that are already running. Once configured, these policies can generally control scheduling without further administrator intervention, simplifying grid management and user job submission while transparently supporting site usage decisions.

About the Author

Charu Chaubal is an engineer in the Grid Computing Engineering group at Sun Microsystems, Inc. He is currently working on grid management and infrastructure software solutions, as well as the prototyping and development of new grid technologies. Frequently called upon as a grid expert for customer engagements and industry events, he has also developed and delivered training courses on grid computing to a variety of customers and partners in the United States and abroad. Charu received a Bachelor of Science in Engineering from the University of Pennsylvania, and a Ph.D. from the University of California at Santa Barbara, where he studied the numerical modeling of complex fluids. He is the author of numerous publications and patents, including several in the field of grid computing.

Acknowledgements

The author would like to recognize the following individuals for their contributions to this article:

- Andreas Haas, Sun Microsystems, N1 Systems
- Carlo Nardone, Sun Microsystems, Client Solutions

References

N1 Grid Engine 6 Administration Guide, Part Number 817-5677-10.

To access this document online, go to <http://docs.sun.com/app/docs/doc/817-5677>

N1 Grid Engine 6 User's Guide, Part Number 817-6117-10.

To access this document online, go to <http://docs.sun.com/app/docs/doc/817-6117>

Chaubal, Charu. "Sun Grid Engine, Enterprise Edition — Software Configuration and Use Cases," *Sun BluePrints OnLine*, July 2003.

To access this document online, go to <http://www.sun.com/blueprints/0703/817-3179.pdf>

Bulhoes, Byun, Castrapel, and Hassaine. "N1 Grid Engine 6 Features and Capabilities," *SUPERG*, Phoenix, Arizona, May 2004. To access this document online, go to:

<http://www.sun.com/products-n-solutions/edu/whitepapers/pdf/N1GridEngine6.pdf>

Ordering Sun Documents

The SunDocsSM program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is

`http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at:

`http://www.sun.com/blueprints/online.html`

