USING HOST GROUPS AND CLUSTER QUEUES IN THE SUN N1™ GRID ENGINE 6 SYSTEM

Charu Chaubal, N1 Systems

Sun BluePrints™ OnLine — August 2005

Part No 819-3165-10
Revision 1.0, 8/1/05
Edition: August 2005

Please
Recycle

Table of Contents

Using Host Groups and Cluster Queues in the Sun N1™ Grid Engine 6 System

Grid engine technology is currently used to power thousands of grids, collections of network-connected servers, providing more efficient use of computing resources. The 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 discusses abstracting collections of resources within the N1 Grid environment using cluster queues and host groups, and explains how these features can be used to simplify administration and implement scheduling policies. The article addresses the following topics:

This article assumes that the reader has a basic understanding of the N1 Grid Engine system and the services it provides. Please see the *N1 Grid Engine 6 Administration Guide* and the *N1 Grid Engine 6 User’s Guide* for more detailed information on using N1 Grid Engine 6 software.

Host Groups

Host groups, introduced in the N1 Grid Engine 6 software release, provide a way to use a single name to refer to multiple hosts. Administrators and users can use host groups to easily reference an arbitrary group of hosts, both for administration purposes as well as for job requirement specification. This additional abstraction mechanism greatly simplifies grid administration and also enables more sophisticated scheduling decisions.

A host group has just two parameters: the host group name and the list of members. The host group name must always begin with the "@" sign. The members can be either individual hosts or other host groups. Some example host groups are given in Table 1.

Table 1. Example host groups.

Name	Members
@sol-sparc64	Hostnames of all 64-bit SPARC systems running the Solaris™ Operating System
@sol-x64	Hostnames of all Opteron (64-bit) systems running the Solaris Operating System
@solaris	@sol-sparc64, @sol-x64
@v20z	Hostnames of all Sun Fire™ V20z servers, regardless of operating system
@deptA	Hostnames of all systems that are owned by Department A
@WestCoastDivision	@deptA, @deptB, @deptC

Benefits of Using Host Groups

Two properties make host groups particularly useful when managing large grid environments:

- *Host groups can be hierarchical.* A host group can contain other host groups as members, which can in turn contain still other host groups. There is no limit to the nesting.
- *Host groups can be overlapping.* A host can belong to more than one host group. In addition, a host group itself can belong to more than one host group.

By defining host groups appropriately, the configuration of execution hosts (hosts that run jobs in the N1 Grid system) becomes greatly simplified. In a large grid, for example, host groups could be created on the basis of operating system, CPU count, or system type. Whenever a new host is added to the grid, it only need be registered in whatever host groups are valid for that host—for example, operating system type and CPU count. Then, the host automatically becomes a member of all upper-level host groups in which the lower level host groups are members. With a well-defined hierarchy, hundreds of hosts of vastly different types can be aggregated in a logical and effective manner.

Managing Host Groups

The following examples illustrate operations commonly executed while managing host groups in an N1 grid environment.

- *Creating host groups*

  Host groups can be set up using the graphical user interface (QMON) or the command line interface. The graphical user interface, shown in Figure 1, enables administrators to create a new host group and then interactively add each execution host.

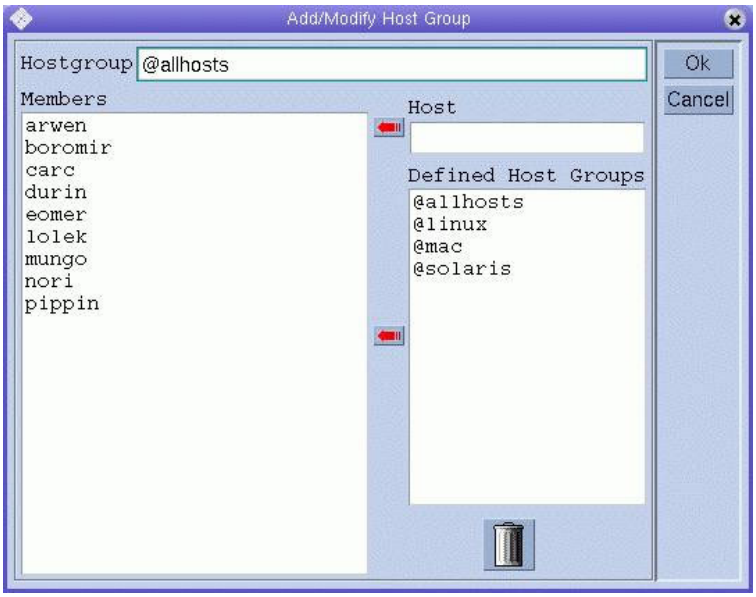

Figure 1. The graphical user interface, QMON, can be used to create host groups.

  For environments with large numbers of hosts, it may be preferred to use the command line interface to create host groups. The administrator can create a text file containing the name of the host group and the list of member hosts, and then use the `qconf` command to create the host group:

```
# qconf -ahgrp filename
```

- *Adding a new host*

  When a new host is added to the grid, administrators need only add it to the base level host groups—it is automatically included in any higher-level derived host groups. Using the example host groups listed in Table 1, assume a new 64-bit SPARC system running Solaris operating system is added to the grid in Department A. Administrators would add it to the `@sol-sparc64` and `@deptA` host groups. This new system is automatically a member of the `@solaris` and `@WestCoastDivision` host groups as well.

The graphical user interface, shown in Figure 1, or the command line interface can be used to add a new host to an existing host group. The following syntax is used to add a new host using the command line interface:

```
# qconf -aattr hostgroup hostlist new_hostname hostgroup
```

For example, the following command adds the host named `newhost` to the `@solaris` host group:

```
# qconf -aattr hostgroup hostlist newhost @solaris
```

The following command adds two hosts, `wkstn1` and `wkstn2`, and one host group, `deptA`, to the existing host group `@WestDiv`.

```
# qconf -aattr hostgroup hostlist "wkstn1 wkstn2 @deptA" @WestDiv
```

- *Displaying host group configurations*

  Host group configurations, including nested host groups, can be displayed with the `qconf` command. For example, the following command shows the host groups and execution hosts that comprise the host group `@divisionX`:

```
# qconf -shgrp_tree @divisionX
@division X
    @deptA
        wkstn-71
        wkstn-72
    @deptB
        wkstn-73
        wkstn-74
    wkstn-75
    server-218
```

Cluster Queues

Cluster queues, another feature introduced in the N1 Grid Engine 6 software release, provide a way to define various job execution parameters that apply to multiple hosts. A queue[1] in the N1 Grid Engine 6 system can be thought of as a container, or description, for a class of jobs. Queues can span multiple execution hosts, and are therefore sometimes referred to as *cluster queues*. Jobs can be submitted to a specific queue, or the N1 Grid Engine software can automatically choose a queue based on specified job requirements and current load conditions. Throughout its lifetime, a running job is associated with its queue.

Cluster queues contain parameters that describe accessibility to the queue, the job execution environment provided by hosts in the queue, and the eligibility for this queue to run jobs. Accessibility parameters include access lists and calendar windows of availability. The job execution environment is defined by various queue methods (start, suspend, etc.), as well as run-time resource limits. Eligibility for running jobs is affected by settings such as load and suspend thresholds, number of available slots, and availability of any resources tied to the queue.

Cluster Queues Simply Administration and Use

Cluster queues enable users and administrators to work with a group of execution hosts by means of a single queue configuration, simplifying administration and use of the grid resources. Because cluster queues are highly flexible, they support all types of environments, such as large sets of homogeneous hosts or small numbers of heterogeneous SMP servers.

Cluster queues in the N1 Grid Engine software differ from queues in many other resource management products. In those products, a queue is often defined on the basis of priority — for example, there may be a low-priority queue and another queue for high-priority jobs. However, cluster queues in the N1 Grid Engine environment are not meant to organize pending jobs solely by priority. Rather, they are meant to define a type of container or environment in which a job runs. For example, cluster queues may be created for servers in a particular department, for running particular classes of applications, or for all servers of a particular model or system capability. Administrators can set up queues to more easily enforce site usage policies, and users can then employ these queues to easily and efficiently run their jobs.

Cluster Queue Parameters

Each cluster queue is assigned a queue name (`qname`) and member list of execution hosts (`hostlist`). The member list of a cluster queue can include individual hostnames, or host groups. Additional

1. In the previous Sun Grid Engine Enterprise Edition 5.3 release, a queue is defined on a per-host basis. In the current N1 Grid Engine 6 release, queues can span multiple hosts and are therefore referred to as cluster queues. In the new terminology, the old queue that existed on a single host becomes a queue instance, which is the intersection of a cluster queue and a single host.

parameters defined by each cluster queue are categorized as described in Table 2. For a complete listing of all cluster queue parameters, please see the *N1 Grid Engine 6 Administration Guide*.

Table 2. Cluster queue parameters.

Category	Characteristics
Accessibility	Determines when a queue can be used and by whom, and who has control over the queue. Examples: `calendar` (days/times jobs can be scheduled), `user_lists` (approved users), and `xuser_lists` (users denied access).
Job execution environment	Defines the environment in which a job runs, and how the job is affected while running. This includes potentially available parallel and checkpointing environment, generic execution methods, and run-time job limits. Examples: `start_method` (generic start method) and `h_rt` (job's hard run-time limit).
Scheduling eligibility	Regulates conditions under which the queue will accept jobs for running and allow running jobs to continue. Also includes any resources which make the queue eligible for special job types. Examples: `slots` (job slots) and `load_thresholds` (allowable system load threshold).

In addition to a global default value, every parameter in a cluster queue definition can have a per-host (or per-host group) value that overrides the default value for that host (or host group). This allows a queue to have parameters defined that are appropriate for most hosts, but allows exceptions to be made wherever needed, eliminating the need to define a unique queue for every possible combination of parameter values.

One example of a parameter for which a per-host override is useful is `slots`, the maximum number of jobs that a queue can run simultaneously. For typical compute-bound jobs, the number of slots is set to the number of CPUs on the execution host. In a heterogeneous grid, administrators can define a global value for all hosts in a queue, reflecting the most typical system configuration. However, individual hosts which have more or fewer CPUs can be listed explicitly with a different value.

An example queue configuration using per-host attribute values is shown in Table 3. In this example, the number of slots is set to two for all execution hosts in the queue. However, the host `durin` overrides the default and uses an attribute value of one. Similarly, the host `ori` uses an attribute value of eight for this parameter.

Table 3. Example queue configuration using per-host attribute values.

Parameter Name	Description	Value
`qname`	Name of cluster queue	`big`
`hostlist`	List of execution hosts	`balrog`, `durin`, `ori`
`slots`	Job slots	`2`, `[durin=1, ori=8]`

Scenario: Using Host Groups and Cluster Queues

Host groups can be used when defining cluster queues, for easier administration. Once a host group scheme is designed and implemented for a grid, cluster queues can be defined entirely without specifying any particular host. As soon as an execution host is given membership in all relevant host groups, everything about that host is automatically specified, and the host automatically becomes eligible for any job whose resource requirements are satisfied by those host groups. This logical separation between virtual job containers (defined by cluster queues) and physical systems (defined by host groups) enables administration and maintenance of large grids in an effective and scalable manner.

The following example use case illustrates these concepts. Suppose there is an environment which consists of the following types of hosts:

- Two-way Opteron systems (Sun Fire V20z servers), running the Solaris 10 Operating System (Solaris 10 OS), Redhat Linux, or SUSE Linux.
- Eight-way SPARC systems (Sun Fire V880 servers), running the Solaris 10 OS.

One way to set up host groups for this environment is to create one set of *basic* host groups, and use them to build up *derived* host groups.

- *Basic host groups*: All members of these host groups are individual hosts, chosen according to their architecture and operating system.
- *Derived host groups*: These host groups contain only other host groups.

Table 4 contains the list of host groups for this example use case. The `@sol-v20z`, `@rdh-v20z`, `@sus-v20z`, and `@v880` host groups contain individual execution hosts. The remaining three derived hosts groups — `@solaris`, `@linux`, and `@v20z` — contain only other host groups.

Table 4. Basic and derived host groups.

Host Group	Members
@sol-v20z	all Sun Fire V20z servers running the Solaris 10 OS
@rdh-v20z	all Sun Fire V20z servers running Red Hat Linux
@sus-v20z	all Sun Fire V20z servers running SUSE Linux
@v880	all Sun Fire V880 servers
@solaris	@sol-v20z,@v880
@linux	@rdh-v20z, @sus-v20z
@v20z	@rdh-v20z,@sol-v20z,@sse-v20z

Suppose this environment runs a combination of serial and parallel jobs. The parallel jobs make use of Message Passing Interface (MPI) libraries on all platforms, but the parallel environments need to be defined differently for the three operating systems: the Solaris 10 OS (`sol-mpi`), Red Hat Linux (`rh-mpi`), and SUSE Linux (`ss-mpi`). This could be due, for example, to differences in the versions of MPI libraries available for each operating system. Additionally, assume there is a special-purpose prolog script defined

for all jobs, which is the same on all Linux hosts (`prolog_lnx.sh`), but different on hosts running the Solaris OS (`prolog_sol.sh`).

Finally, two cluster queues are defined:

- A *normal queue,* for running most jobs.
- An *immediate queue*, which allows high-priority tasks to preempt normal tasks in order to run. This queue runs only serial jobs, limits the duration of these tasks to one hour, and runs them only on the two-way systems.

Table 5 highlights the relevant parameters in the two queues and shows how they are defined in terms of the host groups. Host groups are used to specify the queue's hostlist parameter. Furthermore, host groups are used to conditionally set the values for cluster queue parameters such as the prolog method run before the job starts (`prolog`), the parallel environment list (`pe_list`), and the maximum number of jobs the queue can run simultaneously (`slots`).

Table 5. Queues defined with host groups.

Queue Parameter		Immediate Queue Value	Normal Queue Value
Name	**Description**		
`Hostlist`	List of hosts	`@v20z`	`@v20z, @v880`
`h_rt`	Hard run-time limit	`01:00:00`	`NONE`
`pe_list`	Parallel environment list	`NONE`	`[@solaris=sol-mpi, @rdh-v20z=rhmpi, @sus-v20x=ss-mpi]`
`prolog`	Prolog method; run before job is started	`[@linux=/global/prolog_lnx.sh, @solaris=/global/prolog_sol.sh]`	`[@linux=/global/prolog_lnx.sh, @solaris=/global/prolog_sol.sh]`
`Slots`	Job slots	`[@v20z=2]`	`[@v20z=2,@v880=8]`
`subordinate_list`	List of queues whose jobs are suspended	`normal`	`NONE`

- With the queues and host groups thus defined, the relationship between systems and the job execution container is fully specified, providing greater ease of administration. Although this example use case is small, the concrete advantages are apparent for large, heterogeneous grids.
- Whenever a new host is added to the grid, it only need be added to one of the basic host groups. It immediately acquires all relevant N1 Grid configurations automatically (and immediately starts accepting jobs, unless purposefully disabled).
- If a configuration change is made, such as a newer version of a parallel environment, the change needs to be made only in the queue definition. The change is automatically reflected on the relevant systems.

Scenario: Job Scheduling with Host Groups

Host groups can also be useful when jobs need to run on a particular set of systems or if a particular set of systems is preferred when running certain jobs. The N1 Grid Engine software automatically picks the most suitable and available host for a job, so that the user does not need to know or care about it. However, there could be situations in which the site policy dictates that certain systems are preferred or explicitly must be used. The abstraction layer provided by host groups makes for easier administration and user transparency.

Consider the following use case. Suppose there is an environment with many departments, many users with their own workstations in each department, other resources owned by each department, and a common pool of shared resources owned by the central IT department. The IT policy for this site says that:

- Users typically run their job on their own workstations.
- If there is a job already running, or the workstation is in use for interactive work, the job should run on the pool of servers owned by that user's department.
- If the department pool is full of jobs, the job should run on the site-wide common pool of servers, according to policy shares.
- If the common pool is full of jobs, and some other department pools are under utilized, the job should run on another department's servers.

In this scenario, host groups can be used in combination with soft requests to set preferences for where jobs should be run.

Defining Host Group Configuration

The first step in configuring this environment is to define appropriate host groups. For simplicity, assume only two departments, each with their own servers (Figure 2). In addition, some servers belong to a common pool of shared resources .

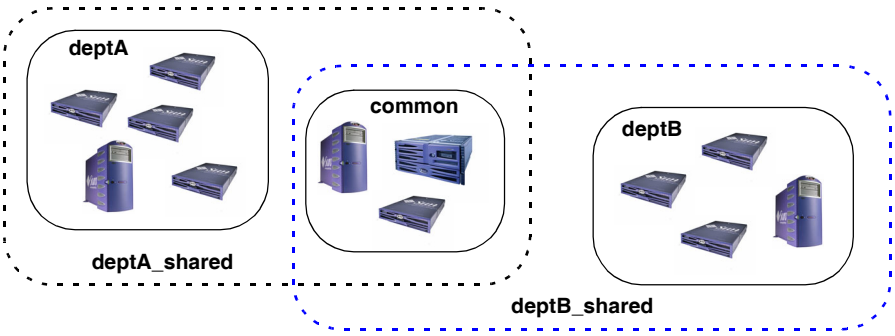

Figure 2. Host groups are created for servers in two separate departments as well as a set of common shared servers.

In this example, both basic and derived host groups are employed. First, basic host groups are created, one for each department (`deptA` and `deptB`) and a third for the set of common shared servers (`common`). Next, two derived host groups are created, one for the combined set of servers that can be used by the first

department (`deptA_shared`) and another for the combined set of servers that can be used by the second department (`deptB_shared`). This example host group scheme is detailed in Table 6:

Table 6. Host group definitions for example use case scenario.

Host Group	Members
@deptA	All servers belonging to Department A
@deptB	All servers belonging to Department B
@common	All servers in the common pool of shared resources
@deptA_shared	@deptA, @common
@deptB_shared	@deptB, @common

Requesting Queues During Job Submission

In order to take advantage of host groups for job submissions, the expanded syntax for requesting queues in the N1 Grid Engine software is used. The `-q` flag to the `qsub` command allows the user to specify a *queue*, *queue instance*, or *queue domain* where a job should run:

- *Queue*

  If a queue is specified, it is selected without preference to any particular execution host in the queue. As long as a host can be found in the hostlist, either directly or by virtue of membership in a host group (or nested host group), that host becomes eligible to run the job. The syntax for such a request is:

```
# qsub -q myqueue job
```

- *Queue instance*

  A queue instance indicates the specific instantiation of a queue on the specified host. In this case, the job runs only on the specified host. The syntax for such a request is:

```
# qsub -q myqueue@hostname job
```

- *Queue domain*

  A queue domain indicates any queue instance which is found on a host in the specified host group. The syntax in this case is:

```
# qsub -q myqueue@@hostgroup job
```

  In all three types of queue requests (queue, queue instance, and queue domain), a regular expression can be used for the queue name to match multiple queues. A single asterisk character indicates a string wildcard, so that any queue satisfies the requests. This in effect makes the request dependent upon only the host or host group specified, and not on the queue name. (When using a regular expression, the specification must be surrounded by single quotes in order to prevent shell expansion of the

expression.) So, for example, to request any host in a particular host group, without regard to the queue, the request syntax is:

```
# qsub -q '*@@hostgroup' job
```

Using Soft Requests

Soft requests are treated as advisory statements by the N1 Grid Engine software, indicating a preference for where a job should run. If there are multiple soft requests, the N1 Grid Engine scheduler attempts to find the host which satisfies as many soft requests as possible. If none of the soft requests can be satisfied by any available job slot, the job is still dispatched. This is in contrast to the normal *hard request*, which must be satisfied by a queue instance or else the job cannot run there.

Soft requests have the exact same syntax as regular resource requests, and are specified at submit time by putting the `-soft` flag in front of all the soft requests. To indicate that any further following requests are hard, the `-hard` flag is inserted before listing them.

Now, using these principles, a user in Department A, who uses a workstation called `sunblade1`, would submit jobs using the following requests:

```
# qsub -soft -q '*@sunblade1' '*@@deptA' '*@@deptA_shared' -hard <optional hard requests> job
```

These soft request options have the following effect:

- The first soft request (`*@sunblade1`) gives preference to the user's own workstations.
- The second soft request (`*@@deptA`) gives preference any host in the Department A host group.
- The third soft request (`*@@deptA_shared`) gives preference to any host in either the Department A host group or the common pool.

The N1 Grid Engine software uses soft requests to rank the list of candidate queue instances where a job should run (the candidate list being all those which satisfy all hard requests). Every satisfied soft request adds one to the ranking. The final ranking based upon this submission statement is described in Table 7. An implicit assumption here is that the user's workstation is always a member of the department pool hostlist.

Table 7. Job ranking based on soft request.

Host	Soft Request Ranking	Reason
Servers belonging to department B	0	Does not satisfy any of the soft requests
Servers belonging to the common pool	1	Satisfy only the soft request for @depA_shared
Servers belonging to department A	2	Satisfy two soft requests: @deptA and @deptA_shared
The user’s workstation	3	Satisfies all three soft requests

Thus, the desired job scheduling preference is satisfied by this scheme. In order to simplify usage for the user, the user-based `sge_requests` file can be used. In each user's home directory, create the file `~/.sge_requests` with the following contents:

```
-soft -q '*@hostname' '*@@deptA' '*@@deptA_shared' -hard
```

where `hostname` is the name of the user's workstation. This specification is then automatically appended to all job requests made by that user. This simplifies job submission for users, and transparently directs their jobs to the preferred servers.

Once again, host groups also simplify the administration of this setup. Any time a department acquires a new server, the hostname can simply be added to the department hostlist, and it automatically participates in the selection scheme. The same is true if the central IT department installs new servers as well. No other configuration is needed to maintain this preference scheme.

Implementing Strict Access Control

The configuration outlined in the previous section indicates only resource *preference*, and not any kind of strict limitation. If there is a need for strict access controls in the N1 Grid, then access lists can be used. For each configured cluster queue, the parameter `user_list` would be modified to indicate which groups have access to which host group. For example, consider the following setting:

```
user_list NONE,[@deptA=deptA_users,@deptB=deptB_users]
```

This indicates that no general restriction is imposed for use of this queue. However, for systems that are part of the `@deptA` host group, only users in the `deptA_users` user list can access them. Similarly, only users in the `deptB_users` user list can access systems in the `@deptB` host group. Any soft request for a job which involves a host group restricted for a user has no effect, since it can never be satisfied.

Since host groups have no inherent userlists themselves, this configuration must be done for every queue of which the host group is a member. Although individual execution hosts in the N1 Grid can have access lists, managing access in the queue configuration preserves the ability to take advantage of the abstraction and grouping afforded by host groups.

Summary

Host groups and cluster queues, new features introduced in the N1 Grid Engine 6 software release, can be used to streamline administration and help implement scheduling policies in the N1 Grid system. Host groups, the ability to reference multiple hosts with a single name, simplify grid administration by enabling large numbers of hosts of potentially vastly different types to be aggregated in a logical and effective manner. Cluster queues provide a means to define job execution requirements that apply to multiple hosts. Having a single queue configuration simplifies administration while still providing the flexibility to set per-host or per-host groups parameters as needed. In addition to simplifying administration, host groups and

cluster queues can be used to help implement site usage policies, making it easy for users to transparently submit jobs to preferred sets of execution hosts.

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`

Ordering Sun Documents

The SunDocs℠ 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`

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN **Web** sun.com