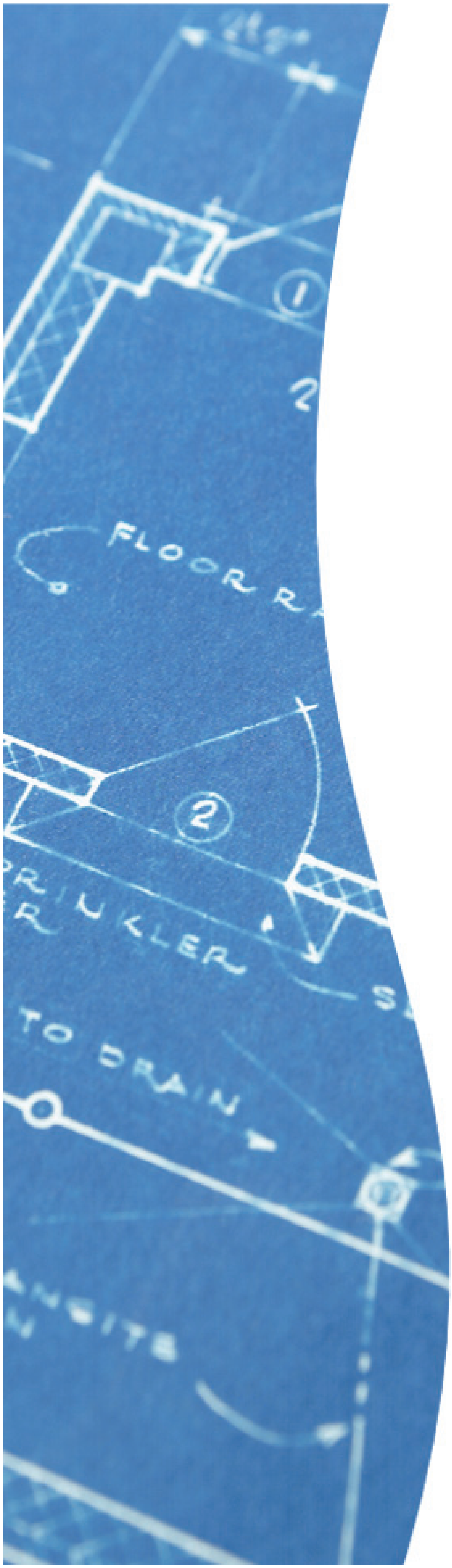

SUN N1™ GRID ENGINE SOFTWARE AND THE **TOKYO INSTITUTE OF TECHNOLOGY SUPERCOMPUTER GRID**

Minoru Hamakawa, Sun Services, Japan

Sun BluePrints™ On-Line — June 2007

Part No 820-1695-10
Revision 1.0, 5/23/07
Edition: June 2007

Table of Contents

Chapter 1
The Tokyo Institute of Technology Supercomputer Grid

One of the world's leading technical institutes, the Tokyo Institute of Technology (Tokyo Tech) created the fastest supercomputer in Asia, and one of the largest outside of the United States. Using Sun x64 servers and data servers deployed in a grid architecture, Tokyo Tech built a cost-effective, flexible supercomputer that meets the demands of compute- and data-intensive applications. Built in just 35 days, the TSUBAME grid includes hundreds of systems incorporating thousands of processor cores and terabytes of memory, and delivers 47.38 trillion[1] floating-point operations per second (TeraFLOPS) of sustained LINPACK benchmark performance and 1.1 petabyte of storage to users running common off-the-shelf applications. Based on the deployment architecture, the grid is expected to reach 100 TeraFLOPS in the future.

This Sun BluePrints™ article provides an overview of the Tokyo Tech grid, named TSUBAME. The third in a series of Sun BluePrints articles on the TSUBAME grid, this document provides an overview of the overall system architecture of the grid, as well as a detailed look at the configuration of the Sun N1™ Grid Engine software that makes the grid accessible to users.

Note – High performance computing environments, like the TSUBAME grid, constantly grow and change. The latest system configuration information and performance characteristics of the TSUBAME grid can be found on the TOP500 Supercomputer Sites Web site located at *http://www.top500.org*, or the Web site of the Global Scientific Information and Computing Center at the Tokyo Institute of Technology located at *http://www.gsic.titech.ac.jp/index.html.en*

The TSUBAME Supercomputer Grid Architecture

Unlike dedicated supercomputers based on proprietary architectures, the TSUBAME supercomputer grid utilizes standard off-the-shelf hardware components to create a high performance computing (HPC) cluster solution. A single Sun Fire™ x64 system architecture is used across 655 servers to power three different types of clusters within the grid. All systems in the grid are interconnected via InfiniBand technology, and are capable of accessing 1.1 petabyte (PB) of hard disk storage in parallel. Incorporating technology from ClearSpeed Technology, Inc., ClusterFS, and Voltaire, as well as the Sun N1 System Manager and Sun N1 Grid Engine software, the TSUMBAME grid runs the Linux operating system to deliver applications to users and speed scientific algorithms and data processing.

1.TOP500 Supercomputing Sites, November 2006, http://www.top500.org/lists/2006/11

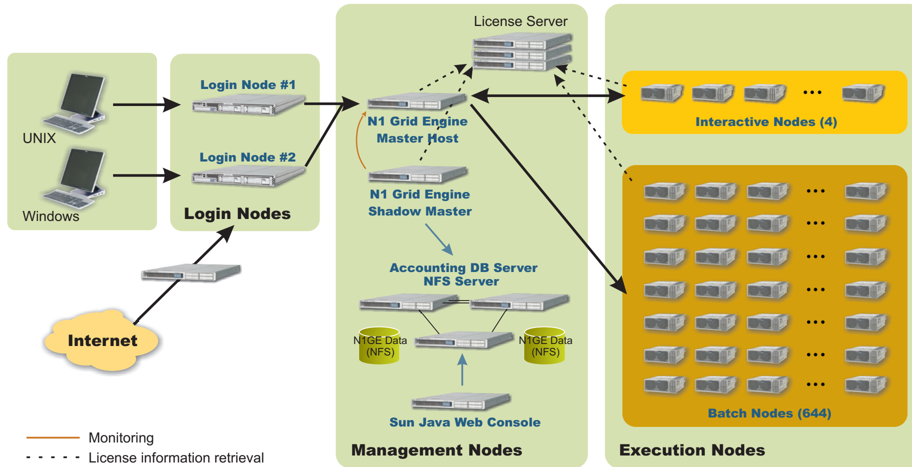

Figure 1-1. The TSUBAME supercomputer grid architecture

The TSUBAME supercomputer grid architecture consists of the following key components:

- *Login nodes*
  Users login to these nodes in order to gain access to the TSUBAME grid. Login access is load balanced among the servers using a round-robin policy. Once logged in, user sessions are transferred automatically to an interactive node by the Sun N1 Grid Engine software.
- *Interactive nodes*
  Four Sun Fire X4600 servers work as interactive nodes in the TSUBAME grid. These servers are utilized by users to create and submit jobs, and compile and run applications.
- *Batch nodes*
  Over 650 Sun Fire X4600 servers work as batch nodes in the TSUBAME grid. All jobs submitted by users — including batch jobs, as well as graphical user interface and command line applications such as PATRAN, Mathematica, and MATLAB — run on these execution nodes. Access to all batch nodes is managed by the Sun N1 Grid Engine software. Users cannot access these nodes using remote login tools, such as ssh, rsh, or telnet.
- *Management nodes*
  Management nodes include master and shadow master hosts for the Sun N1 Grid Engine software, as well as the HA-NFS server containing Sun N1 Grid Engine software configuration data, the accounting information from HA-PostgreSQL, and a licensing server, such as FLEXlm. Sun Fire X4100 servers with InfiniBand connections are used for the Sun N1 Grid Engine software mast and shadow systems, as well as licensing, LDAP, and backup servers. Sun Fire X4100 servers with Ethernet connections are used for the NFS server, database server, and Sun N1 System Manager software.

Built-In Redundancy

In order to support the availability of the TSUBAME grid and ensure users can login to the system and submit jobs, Sun Cluster 3.1 software provides HA-NFS capabilities to protect the availability of Sun N1 Grid Engine configuration data and the shadow master. Because unexpected downtime of the database server does not have a direct affect on users, Sun N1 Grid Engine software writes database information to a flat file on the NFS server before the Sun N1 Grid Engine software places it into the database. In the event access to the database is interrupted, the database can be reloaded with the information stored in the file on the NFS server.

Chapter 2

Sun N1™ Grid Engine Software Configuration

This chapter describes the configuration details of the Sun N1 Grid Engine software. Not intended as a comprehensive configuration reference, this chapter describes only the non-default configuration settings for the Sun N1 Grid Engine software. More information on configuring the Sun N1 Grid Engine software can be found in the *Sun N1 Grid Engine Installation Guide* located at *http://docs.sun.com*

- Global execution node configuration (`qconf -se global`)
- Complexes for execution nodes (`qconf -se hostname`)
- Execution node configuration (`qconf -sconf hostname`)
- Scheduler configuration (`qconf -ssconf`)
- Parallel environment configuration (`qconf -sp pe-name`)
- Queue configuration (`qconf -sq queue_name`)

Configuring the Global Execution Node

The `qconf -se global` command provide configuration information for the global execution node. A portion of the command output is shown below.

```
complex_values batch_sgeadmin=32,batch_sgeuser=32,batch_nec=32,
.....
max_cpu_admin=64,max_cpu_bes1=128,max_cpu_bes2=1024, ¥
.....
```

In the Sun N1 Grid Engine software, a *complex* is a set of resource attribute definitions that can be associated with a queue, host, or the entire cluster. The TSUBAME grid uses the following complexes as the global consumable complex to limit the maximum number of CPUs users and groups can use:

- batch_*user_name*, the maximum number of CPUs a user can use at a given time
- max_cpu_*group_name*, the maximum number of CPUs a group can use at a given time

The TSUBAME grid uses a wrapper command (`n1ge`) for all `qsub` and `qrsh` commands. Three types of queues are used.

- General queue, the default queue that is free to use (`high`, `default`)
- Best effort queue, a queue that is charged at a flat rate (`bes1`, `bes2`)
- Service level agreement queue, a queue that is charged on a pay for usage basis (`sla1`, `sla2`). The TSBUBAME grid uses the accounting information provided by the Sun N1 Grid Engine software to calculate the fee for each group.

For users that do not have authority to use the best effort or service level agreement queues, the TSUBAME grid limits the maximum number of CPUs users can access at any given moment by using the `n1ge` wrapper command which automatically specifies the `-l batch_`*`user_name`*`=1` option to the `qsub` and `qrsh` commands. For the best effort queue, the TSUBAME grid limits the maximum number of CPUs the group can use at the same time by using the `n1ge` wrapper command, which specifies the `-l max_cpu_`*`group_name`*`=1` option. In addition, Access Control Lists (ACL) are also used for these queues, by group.

Execution Node Complexes

The TSUBAME grid uses two complexes as the consumable complex for execution nodes within the grid.

- *h_slots*, the number of CPUs on execution nodes
- *use_mem_size*, the size of available memory on execution nodes, in GB

```
complex_values        h_slots=16,use_mem_size=32
```

The maximum number of CPUs used by jobs in a queue instance at the same time, such as `queue_name@nodename`, is limited by the number of slots in the Sun N1 Grid Engine software. Because an execution node can belong to several Sun N1 Grid Engine software queues, the following problematic sequence can result in performance problems and node unavailability.

- `NodeA` has four CPUs.
- `NodeA` belongs to both `queueA` and `queueB`, and each slot value is 4.
- Four jobs run on `nodeA@queueA`, while three jobs run on `nodeA@queueB`.
- The total number of jobs on `nodeA` is seven.

To resolve this problem, `h_slots` complexes can be created on the execution nodes, as described in Table 2-1. It is important to note the setting `requestable=FORCED`. In addition, the maximum of CPUs used by jobs at the same time becomes the value of `h_slots` by setting `h_slots` on all execution hosts. Of course, `h_slots` should be set to the number of CPUs in the system. Such a configuration ensures the problem identified above does not occur. To run more jobs than the number of CPUs on the execution node, change the value of `h_slots` appropriately. Do not forget to set the value of `slots` to the same value as `h_slots`. In the TSUBAME grid, `h_slots` is set to 1,000 for all interactive nodes.

Table 2-1. h_slots Settings

Name	Shortcut	Type	Relop	Requestable	Consumable	Default	Urgency
h_slots	h_slots	INT	<=	Forced	Yes	0	0

Another issue can arise over memory. The `use_mem_size` parameter is used for memory reservation. The Linux operating system kernel can hang if more memory is used than is available on the system. To mitigate this concern, the TSUBAME grid employs the following preventive measures. However, both `h_slots` and `use_mem_size` are specified by the wrapper command to the `qsub` and `qrsh` commands, rather than directly by users.

- The wrapper command specifies the size of memory for a job to `use_mem_size` automatically. In addition, the Sun N1 Grid Engine scheduler allocates the job to the execution node on which the specified memory is available.
- If a job will use more memory than the value of the `use_mem_size` parameter, the Sun N1 Grid Engine software kills the job immediately.

Configuring Execution Nodes

The following example describes the configuration of execution nodes. A key aspect of the configuration is its use of `ssh` and `sshd` rather than `rsh`. More information on the use of `ssh` can be found in “SSH for Sun N1 Grid Engine Software” on page 13.

```
mailer                        /bin/mail
xterm                         /usr/bin/X11/xterm
qlogin_daemon                 /usr/local/sge-ssh/sbin/sshd -i
rlogin_daemon                 /usr/local/sge-ssh/sbin/sshd -i
rsh_daemon                    /usr/local/sge-ssh/sbin/sshd -i
rsh_command                   /usr/bin/ssh -t -X
```

Configuring the Scheduler

The scheduler configuration for the TSUBAME grid is detailed below. Several items are important to note:

- The TSUBAME grid does not use the `load_adjustments_decay_time` and `job_load_adjustment` parameters in order to reduce the load on the scheduler.
- The load_formula parameter is set to load-h_slots, ensuring the scheduler allocates jobs to the execution node with the most unused CPUs. If more than one node has the same amount of CPUs idle, the scheduler allocates the job to the node with the lightest CPU load. Note the TSUBAME grid does not allocate jobs to the node with the minimum job load. By making its decision based on CPU usage instead, the grid is better able to take into account application usage characteristics. In fact, it is difficult for the scheduler to optimize CPU utilization if the execution nodes are allocated based on node job load.

```
algorithm                          default
schedule_interval                  0:0:15
maxujobs                           0
queue_sort_method                  load
job_load_adjustments               NONE
load_adjustment_decay_time         0:0:0
load_formula                       np_load_avg-h_slots
schedd_job_info                    true
flush_submit_sec                   0
flush_finish_sec                   0
params                             none
reprioritize_interval              0:0:0
halftime                           168
usage_weight_list                  cpu=1.000000,mem=0.000000,io=0.000000
compensation_factor                5.000000
weight_user                        0.250000
weight_project                     0.250000
weight_department                  0.250000
weight_job                         0.250000
weight_tickets_functional          0
weight_tickets_share               0
share_override_tickets             TRUE
share_functional_shares            TRUE
max_functional_jobs_to_schedule    200
report_pjob_tickets                TRUE
max_pending_tasks_per_job          50
halflife_decay_list                none
policy_hierarchy                   OFS
weight_ticket                      0.010000
weight_waiting_time                0.000000
weight_deadline                    3600000.000000
weight_urgency                     0.100000
weight_priority                    1.000000
max_reservation                    200
default_duration                   0:30:0
```

Parallel Environment

With many applications supporting parallelization techniques, such as the Message Passing Interface (MPI), OpenMP, TCP Linda and the Distributed Data Interface (DDI), the parallel environment is a critical component of the TSUBAME grid. To date, the TSUBAME grid supports 33 parallel environments, including the popular Voltaire MPI. The example below details the configuration of the parallel environment in the TSUBAME grid.

```
pe_name           vol_mpi_8p
slots             20000
user_lists        NONE
xuser_lists       NONE
start_proc_args   /n1ge/TITECH_GRID/tools/pe/voltaire_mpi/startmpi.sh \
                  -catch_rsh  $pe_hostfile
stop_proc_args    /n1ge/TITECH_GRID/tools/pe/voltaire_mpi/stopmpi.sh
allocation_rule   8
control_slaves    TRUE
job_is_first_task FALSE
urgency_slots     min
```

Configuring the Queues

The TSUBAME grid architecture utilizes several queues (Figure 2-1). Resources are not limited in the queue configuration, with the exception of the core file size.

- The *interactive queue* is free to use (`high`, `default`). All users are allocated to the interactive queue upon login to the TSUBAME grid.
- The *default queue* is used by all nodes connected to the same edge InfiniBand switch, and is free to use.
- The *best effort services queues* (`bse1`, `bse2`) is a flat-rate queue that is used by nodes connected to the same InfiniBand switch. Pricing is based on the number of CPUs a group is allowed to access at the same time. Only qualified users can access these queues. The number of CPUs users can use simultaneously is limited to the number of CPUs registered to the queue. Users with access to these queues can also use the default, high, mopac, avs, and sas queues.
- The *service level agreement queues* (`sla1`, `sla2`) are charged on a usage basis and aim to provide guaranteed service performance to users. Pricing for each group is calculated as the runtime of all jobs multiplied by the number of node multiplied by the price per hour. All nodes that access these queues are connected to the same edge InfiniBand switch. Only qualified users can access these queues. Because the average load threshold tends to be low, users can monopolize nodes. No limitations are placed on the amount of CPU resources available to jobs. In fact, users can use 120*16=1920 CPUs/job. Users with access to these queues can also use the default, high, mopac, avs, and sas queues.
- The *mopac queue* is free to use. A specific queue for mopac is included because of node locked licensing.
- The *avs queue* is free to use.
- The *sas queue* is free to use. A specific queue for sas is included because of node locked licensing.

Users without access to the best effort services or service level agreement queues can use the default, high, mopac, avs, and sas queues.

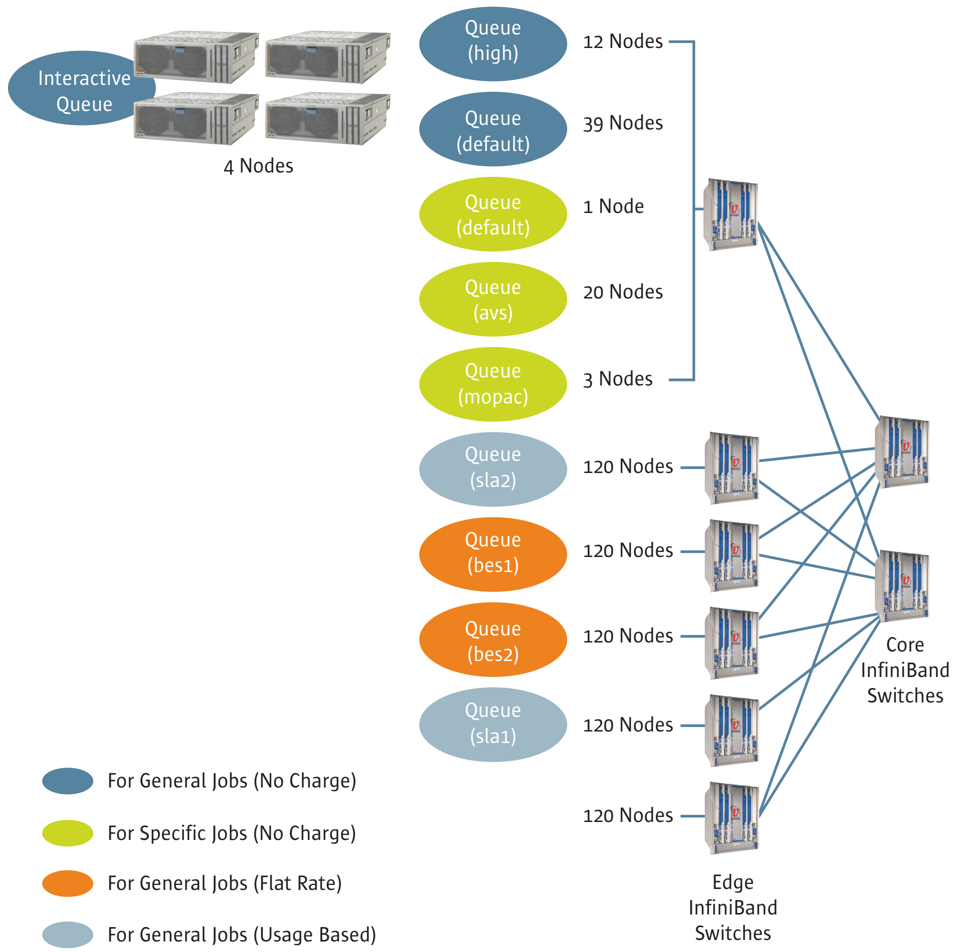

Figure 2-1. Queue configuration in the TSUBAME grid

Table 2-2 summarizes the configuration of the queues in the TSUBAME grid.

Table 2-2. Queue Configuration

Queue Name	Number of Nodes	load_avg Limit	Jobs Per Node Limit	Other Limits
interactive	4	28	1000	4 parallelization 4 GB/process (soft)
high	12	16	16	
default	39	16	16	Less than 32 CPUs/job
bse1, bse2	120	16	16	
sla1, sla2	120	1	16	
mopac	3	28	16	
avs	20	28	16	
sas	1	28	16	

Following is an example of the queue configuration in the TSUBAME grid.

```
qname                 bes1
hostlist              @B01 @B02 @B03 @B04 @B05 @B06 @B07 @B08x
seq_no                0
load_thresholds       np_load_avg=1.00
suspend_thresholds    NONE
nsuspend              1
suspend_interval      00:05:00
priority              0
min_cpu_interval      00:05:00
processors            UNDEFINED
qtype                 BATCH INTERACTIVE
ckpt_list             NONE
pe_list               make vol_mpi_rr vol_mpi_fillup nastran_rr nastran_fillup \
                      abaqus_smp MP molpro vol_mpi_1p vol_mpi_2p vol_mpi_4p \
                      vol_mpi_8p vol_mpi_16p vol_mpi_2t vol_mpi_4t vol_mpi_8t \
                      vol_mpi_16t openmpi_rr openmpi_fillup openmpi_1p \
                      openmpi_2p openmpi_4p openmpi_8p openmpi_16p linda_8p \
                      linda_4p linda_2p linda_1p ddi_16p ddi_8p ddi_4p ddi_2p \
                      ddi_1p
rerun                 TRUE
slots                 16
tmpdir                /tmp
shell                 /bin/csh
prolog                /n1ge/TITECH_GRID/tools/util/queue_prolog
epilog                /n1ge/TITECH_GRID/tools/util/queue_epilog
shell_start_mode      posix_compliant
starter_method        /n1ge/TITECH_GRID/tools/util/job_starter
suspend_method        /n1ge/TITECH_GRID/tools/util/suspend_method
resume_method         /n1ge/TITECH_GRID/tools/util/resume_method
terminate_method      NONE
notify                00:00:60
owner_list            NONE
user_lists            permit_bes
xuser_lists           NONE
subordinate_list      NONE
complex_values        small_job=TRUE
projects              NONE
xprojects             NONE
calendar              NONE
initial_state         default
s_rt                  INFINITY
h_rt                  INFINITY
s_cpu                 INFINITY
h_cpu                 INFINITY
s_fsize               INFINITY
h_fsize               INFINITY
s_data                INFINITY
h_data                INFINITY
s_stack               INFINITY
h_stack               INFINITY
s_core                0
h_core                0
s_rss                 INFINITY
h_rss                 INFINITY
s_vmem                INFINITY
h_vmem                INFINITY
```

The queue_prolog and job_starter tasks help manage and monitor jobs and queues.

- *queue_prolog*
  The queue_prolog task adds accounting information to a job so that job failures can be analyzed. The additional information includes the submit hostname, the shell used, the directory from which the user submitted the job, the job command and arguments, the job type (qsub or qrsh), and any output and error files created by the Sun N1 Grid Engine software. In addition, the queue_prolog task checks if an application license is available through use of the lmutil lmstat command, when the application uses FLEXlm as the license server. If a job cannot obtain the requested license, the queue_prolog task exits with a return value of 99, and the Sun N1 Grid Engine software automatically resubmits the job to the queue.
- *job_starter*
  The job_starter task monitors the standard output and errors for jobs. If error messages occur, the job_starter task exits with a value of 99, and the Sun N1 Grid Engine software automatically resubmits the job to the queue. It is important to note the job_starter task executes the ulimit -v *xx* command to limit the memory size for a job.

Key Challenges and Implementation Policies

Implementation of the TSUBAME grid posed challenges that resulted in the creation of several policies.

- Accounting — Tokyo Tech required the ability to obtain the exact amount of CPU and memory resources used by each job, even jobs in the interactive queue. This requirement would not pose a challenge if the rsh capabilities of the Sun N1 Grid Engine software did not have missing prompt and rsh session limit issues. As a result, ssh is used instead for the Sun N1 Grid Engine software. Details of the ssh configuration can be found in “SSH for Sun N1 Grid Engine Software” on page 13.
- Use of MPI — In order to support a wide variety of applications and users, the TSUBAME grid must provide access to a wide variety of MPI implementations, including Voltaire MPI, MPICH, PGI-MPI, HP-MPI, and OpenMPI. In addition, DDI and TCP Linda are also needed. The behavior of each MPI implementation was evaluated and integrated with the Sun N1 Grid Engine software. While time-consuming, these efforts resulted in more reliable integration. More information on the integration of applications and the Sun N1 Grid Engine software is described in “Integrating Applications with Sun N1 Grid Engine Software” on page 46.
- CPU availability — The TSUBAME grid incorporates 655 Sun Fire X4600 servers, each with 16 processor cores. As a result, the execution nodes in the grid provide access to 10,480 cores. The number of processes is fixed at eight processes per node, or 16 processes per node when $round_robin and $fill_up are not used,

in order to ensure consistent application performance over time. With so many servers running applications, the TSUBAME grid runs MPI jobs with the same edge InfiniBand switch in order to reduce the load on the core InfiniBand switches.

- Parallel applications — While the TSUBAME grid supplies only 26 applications to users, each program uses different parallelization mechanisms. These differences make it difficult to obtain the exact amount of CPU resources used by each application. More information on the strategy employed to deal with these considerations can be found in “Integrating Applications with Sun N1 Grid Engine Software” on page 46.

Chapter 3
SSH for Sun N1 Grid Engine Software

This chapter discusses how to create `ssh` for the Sun N1 Grid Engine software.

Why Use SSH?

Unfortunately, `rsh` has two visible and considerable problems in the Sun N1 Grid Engine software environment.

- Missing prompt — The shell prompt is not displayed for applications, such as MATLAB and Mathematica, when the `pseudo-tty` device is not allocated. This is a serious problem when Electronic Design Automation (EDA) applications are used, because many incorporate a unique shell. Consider the example below. When a job is submitted in this manner, the `bash` prompt is not displayed. However, `ssh` can resolve this issue by using `-t` option. Because obtaining the exact amount of CPU usage for each job is a critical requirement, `ssh` for Sun N1 Grid Engine is used in the grid architecture.

```
$ qrsh bash

id -a
uid=3001(sgeuser) gid=10(staff) groups=10(staff),20090
```

- `rsh` session limits — The rsh utility uses approximately 512 to 1,024 ports, resulting in a theoretical limit of 256 sessions. With 10,480 CPUs in the grid, such a low session limit fails to provide the needed session connectivity. This problem is overcome through the use of `ssh`, as `ssh` can create over 10,480 sessions.

Creating SSH for Sun N1 Grid Engine Software

The following steps outline how to create `ssh` for the Sun N1 Grid Engine software. More detailed information can be found in the mail archives for Sun N1 Grid Engine software on the SunSource.Net site located at gridengine.sunsource.net

1. Download the Sun N1 Grid Engine software source from http://gridengine.sunsource.net. The TSUBAME grid uses version 6.0u7.
2. Download the `ssh`, `ssl`, and `openssh` source code. The TSUBAME grid uses `ssh` 4.3p1 and OpenSSL 0.9.7i.
3. Create the `libuti.a`, `liblck.a`, `librmon.a` and `libsgeremote.a` libraries by compiling the Sun N1 Grid Engine software.
4. Compile `ssh` with the Sun N1 Grid Engine software libraries. If necessary, first compile `ssl`. Be sure to modify *sshd.c*, *session.c*, and the *makefile*. An example modified *sshd.c* file for version 4.3p1 is listed below.

```
/* Re-exec fds */
#define REEXEC_DEVCRYPTO_RESERVED_FD    (STDERR_FILENO + 1)
#define REEXEC_STARTUP_PIPE_FD          (STDERR_FILENO + 2)
#define REEXEC_CONFIG_PASS_FD           (STDERR_FILENO + 3)
#define REEXEC_MIN_FREE_FD              (STDERR_FILENO + 4)

/* N1GE Patch */
#define SGESSH_INTEGRATION

#ifdef SGESSH_INTEGRATION
extern int sgessh_readconfig(void);
extern int sgessh_do_setusercontext(struct passwd *);
#endif

extern char *__progname;
.....
        /* Drop privileges */
        /* do_setusercontext(authctxt->pw); */

        /* N1GE Patch */
        #ifdef SGESSH_INTEGRATION
                sgessh_do_setusercontext(authctxt->pw);
        #else
                do_setusercontext(authctxt->pw);
        #endif

        /* It is safe now to apply the key state */
        monitor_apply_keystate(pmonitor);
.....
#ifdef HAVE_SECUREWARE
        (void)set_auth_parameters(ac, av);
#endif
        __progname = ssh_get_progname(av[0]);
        init_rng();

        /* N1GE Patch */
        #ifdef SGESSH_INTEGRATION
                sgessh_readconfig();
        #endif

       /* Save argv. Duplicate so setproctitle emulation doesn't clobber it */
```

An example modified *session.c* file for version 4.3p1 is listed below. The *session.c* file does not need to be modified unless */etc/nologin* is going to be used on the execution nodes.

```
                 /* When PAM is enabled we rely on it to do the nologin check */
                 if (!options.use_pam)
                         /* N1GE Patch */
                         /* do_nologin(pw); */
                 do_setusercontext(pw);
                 /*
```

Be sure to modify the `LIBS` entry in the Makefile in order to link with the Sun N1 Grid Engine libraries. Note that *../../LINUXAMD64_26* is the directory containing `libuti.a`, `liblck.a,` and `librmon.a`, while *../../../3rdparty/remote/ LINUXAMD64_26* is the directory containing `libsgeremote.a`. If the Makefile includes `-lutil` in the `LIBS` entry, remove *libutil.so* from the *../../LINUXAMD64_26* directory.

```
LIBS=-lcrypto -lz -lnsl -lm -lpthread -lcrypt -lresolv  \
-L../../../../LINUXAMD64_26 \
-L../../../../3rdparty/remote/LINUXAMD64_26/ -lsgeremote -luti \
-llck -lrmon
```

5. Configure the execution nodes.

```
qconf -mconf hostname
```

6. Next, configure the execution nodes to enable the Sun N1 Grid Engine software to use `ssh` and `sshd` for interactive jobs, as described in “Configuring Execution Nodes” on page 6. Remember to specify the `-t` option in the `rsh_command` command line.

```
mailer                        /bin/mail
xterm                         /usr/bin/X11/xterm
qlogin_daemon                 /usr/local/sge-ssh/sbin/sshd -i
rlogin_daemon                 /usr/local/sge-ssh/sbin/sshd -i
rsh_daemon                    /usr/local/sge-ssh/sbin/sshd -i
rsh_command                   /usr/bin/ssh -t -X
```

To test the availability of ssh, perform the following steps:

1. Submit an interactive job, such as `bash`, and confirm the application prompt is present.

```
tgg075002 admin/sun> qrsh bash
sun@tgg075043:~>
```

2. Confirm the GID is allocated by the Sun N1 Grid Engine software using the `id -a` command. If the GID allocated by the Sun N1 Grid Engine software is located, the tight integration of ssh with the software succeeded. In the following example, the GID is 30708.

```
sun@tgg075043:~> id -a
uid=1901(sun) gid=2000(user) groups=1001(katolab),2000(user),30708
sun@tgg075043:~>
```

3. Confirm the ability to obtain the CPU time and memory size of the job by using the Sun N1 Grid Engine software `work` command.

```
sun@tgg075043:~> /n1ge/examples/jobsbin/lx24-amd64/work -f 4 -w 30
Forking 3 times.
sun@tgg075043:~> exit
```

4. Confirm the CPU time for the job with the `qacct` command. In the example below, the `qacct` command reports a CPU time of 120 seconds, as calculated by 4 CPUs multiplied by 30 seconds per CPU.

```
tgg075002 admin/sun> qacct -j XXXXXX
==============================================================
qname        default
hostname     tgg075043
group        user
owner        sun
…………..
…………..
cpu          120
mem          0.569
io           0.000
iow          0.000
maxvmem      85.043M
```

Users migrating from LSF to the Sun N1 Grid Engine software also can take advantage of this solution, as the `rsh` issues exist with LSF and the Sun N1 Grid Engine software as well.

Chapter 4
The Login System

This chapter describes the login system of the TSUBAME grid. Because Tokyo Tech needed to obtain CPU and memory utilization statistics even on interactive nodes, login access to the interactive nodes from client PCs and workstations is denied. Login access is possible only through the Sun N1 Grid Engine software. The initial strategy for the login process involved the following steps:

- Users log in to login nodes.
- Users submitting interactive jobs specify use of the interactive queue. Users submitting batch jobs specify use of the batch queues.

However, Tokyo Tech required users to be transferred to interactive nodes immediately upon logging in to the login nodes. Because it is difficult to treat user logins differently for interactive and batch jobs, a wrapper command is used, called `gridsh`, that handles the login process. The new strategy for the login process involves the following steps:

- Users log in to login nodes.
- The `gridsh` script transfers user sessions to interactive nodes if `telnet` or `ssh` is used.
- Users submit batch jobs on interactive nodes. When users perform other tasks, such as compiling, debugging or running graphical applications, these processes run on interactive nodes.

Figure 4-1 provides an overview of the client access process. Because all execution and login nodes are configured with `ssh` host authentication, users do not need to enter passwords once login authentication is complete.

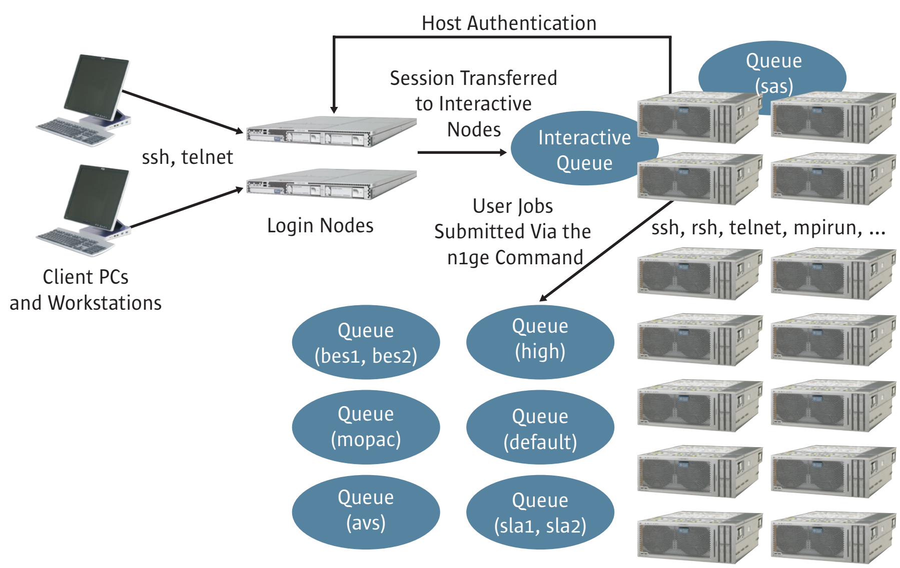

Figure 4-1. The client login process in the TSUBAME grid

Using the Login Process on the TSUBAME Grid

The following examples illustrate the login process on the TSUBAME grid.

```
[sun@media-o ~]$ ssh login
Password:
Forwarding to N1GE Interactive Queue.....
Tue Oct 31 18:29:20 JST 2006
tgg075003 admin/sun> id -a
uid=1901(sun) gid=2000(user) groups=1001(katolab),2000(user),30347
tgg075003 admin/sun> qstat -u sun
job-ID  prior   name   user    state submit/start at     queue slots ja-task-ID
-----------------------------------------------------------------------------------
 342485 0.50500 LOGIN  sun     r     10/31/2006 18:29:18 interactive@tgg075003 1
tgg075003 admin/sun>
```

```
[sun@media-o ~]$ ssh -t login sas
Password:
Forwarding to SAS Node.....
Tue Oct 31 18:31:54 JST 2006
tgg075001 admin/sun> id -a
uid=1901(sun) gid=2000(user) groups=1001(katolab),2000(user),30295
tgg075001 admin/sun> qstat -u sun
job-ID  prior   name   Å@user    state submit/start at     queue @slots ja-task-ID
-----------------------------------------------------------------------------------
 342492 0.50500 LOGIN_SAS  sun   r     10/31/2006 18:31:51 sas@tgg075001 1
tgg075001 admin/sun>
```

Chapter 5
Using the n1ge Command

This chapter describes the `n1ge` wrapper command that interfaces with the `qsub` and `qrsh` commands. The purpose of the `n1ge` command is to:

- Make the supercomputer grid user friendly. Users are unwilling to specify complex commands and options, such as `–l h_slots=1`, `use_mem_size=1`, `FEATURE=`*XX*, `–v` *XX* `–pe` *XX*, and more. To ease this task, the n1ge command automatically specifies options for users and frees them from needing to learn all the options available for the Sun N1 Grid Engine software.
- Make it easy to manage user jobs. Administrators can manage jobs easily by specifying the options to be passed to the `qsub` and `qrsh` commands, as needed.

Overview of the n1ge Command

Figure 5-1 illustrates the behavior of the `n1ge` command. For example, if a user runs the MATLAB application without using the `n1ge` command, MATLAB runs on the interactive node on which the user initiated execution. However, if a user runs MATLAB with the `n1ge` command (`$n1ge matlab`), the application runs on a batch node allocated by the Sun N1 Grid Engine software. As a result, the `n1ge` command makes using the grid easy.

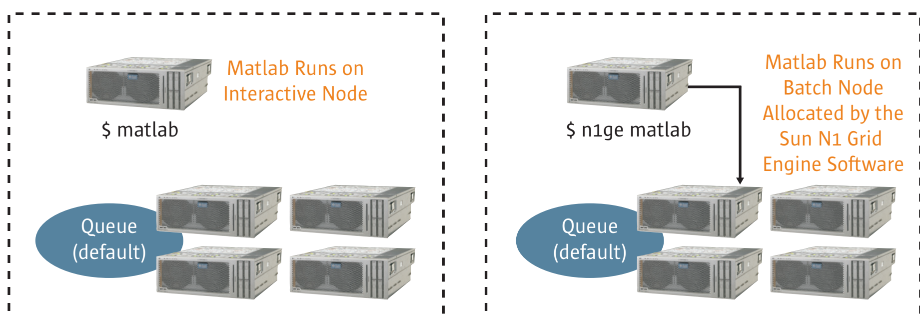

Figure 5-1. The n1ge command automatically allocates jobs to batch nodes in the supercomputer grid

n1ge Command Options

The following output lists the options to the `n1ge` command.

```
n1ge [n1ge_options....] <program> [options...]
    n1ge_options:
        -help         :: This message.
        -verify       :: Do not submit just verify.
        -smp <cpus>   :: Request SMP machines.
                         <cpus> Specify the number of cpus to run on.
        -mpi <cpus>   :: Request using message passing interface.
                         <cpus> Specify the number of cpus to run on.
        -openmpi <cpus>
                      :: Request using message passing interface.(MPI2)
                         <cpus> Specify the number of cpus to run on.
        -linda <cpus> :: Request using linda for Gaussian 03 rev D.02.
                         <cpus> Specify the number of cpus to run on.
        -ddi <cpus>   :: Request using DDI for GAMESS.
                         <cpus> Specify the number of cpus to run on.
        -mem <memory size>
                      :: [MPI Jobs]
                         <memory size> Specify the size of memory for each process of job.(GBytes)
                                       With -mpi, -openmpi, -ddi options.
                         [NonMPI Jobs]
                         <memory size> Specify the size of memory for the job not processes.
                                       ONLY with -smp option.
                         ex1) "-smp 8 -mem 16" means this job will use 16GB and limit 16GB/job.
                        ex2) "-mpi 4 -mem 2" means this job will use 2GB*4=8GB memory and limit 2GB/proc.
        -apps_list    :: Show Application Name you can specify with -apps or -apps_help option.
        -apps <APPS Name>
                      :: Specify Application Name
                         <APPS Name> apps_name[,apps_name,...]
        -apps_help <APPS Name>
                      :: Show applicatoin specific help message of n1ge command.
                         <APPS Name> apps_name
        -q <Queue Name>
                      :: Bind job to the specified queue(s).
                         <Queue Name> queue[,queue,....]
        -mail <Mail Address>
                      :: Request mail notification when begging, ending
                         of the job and aborted or suspended the jobs.
                         <Mail Address> Specify the E-mail Address.
        -env <variable_list>
                      :: Export these environment variables on Execution Hosts.
                         <variable_list> variable[=value][,variable[=value],....]
        -fore         :: Executing this job as foreground job.
        -noreq        :: Define job as NOT Re-queueing.
        -paramfile <filename>
                      :: Specify parameter file of Voltaire MPI.
                         You can get the sample paramfile as /usr/voltaire/mpi/doc/param.sample
                         You need to use -mpi option together.
        -N <JOB NAME> :: Specify Job Name
        -hold_jid <Job-ID list>
                      :: Define jobnet interdependencies
                         <Job-ID list> {job_id|job_name|reg_exp}[,{job_id|job_name|reg_exp},...]
        -sgeout <filename>
                      :: Specify standard output stream path(s) of N1GE
        -g <Group Name>
                      :: Specify the Group Name.
        -rt <runtime(min)>
                      :: Specify the runtime of the job when you belong to SLA group.
                         <runtime(min)> In minutes.
```

Inside the n1ge Command

This section describes how the `n1ge` command works through the presentation of examples. The core capabilities of the `n1ge` command include:

- Creation of options for the `qsub` and `qrsh` commands
- Creation of the job script for the `qsub` and `qrsh` commands

In addition, the `n1ge` command sets options based on the user or group executing the command, in order to place a limit on the maximum number of CPUs that can be used by the job. For example, depending on whether or not the user specifies a group, the results of the `n1ge` command vary. In the examples below, text in bold highlights the differences in the results of the two commands.

Without specifying the group:

```
tgg075001 admin/sun> n1ge id
--
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A "id" -l
h_slots=1,use_mem_size=1.000000,batch_sun=1 -N OTHERS -v
N1GE_MEM=1048576 -q default -r y /home/admin/sun/.sgejob/OTHERS29555.sh
```

With specifying the group:

```
tgg075001 admin/sun> n1ge -g admin id
--
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A "id" -P admin -l
h_slots=1,use_mem_size=1.000000,\
max_cpu_admin=1,small_job=TRUE -N OTHERS -v N1GE_MEM=1048576 -q bes1 \
-r y /home/admin/sun/.sgejob/OTHERS29833.sh
```

It is important to note that users do not need to specify the `-g` option each time, if the `N1GE_GROUP` environment variable is set. The following example shows the output of the `n1ge` command with the `N1GE_GROUP` environment variable set. Note this command is equivalent to the `n1ge -g admin id` command.

```
tgg075001 admin/sun> setenv N1GE_GROUP admin
tgg075001 admin/sun> n1ge -verify id
--
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A "id"  -P admin -l
h_slots=1,use_mem_size=1.000000,\
max_cpu_admin=1,small_job=TRUE -N OTHERS -v N1GE_MEM=1048576 -q bes1 \
-r y /home/admin/sun/.sgejob/OTHERS29984.sh
```

Parallelization Options

The `n1ge` command provides five parallelization options.

-smp Option

The `-smp` option is designed for symmetric multiprocessing (SMP) jobs, such as OpenMP, pthread, and script-level parallelization, which utilize multiple CPUs on a single execution node. The Sun N1 Grid Engine software sets the value of the `allocate_rule` parameter of the MP parallel environment to `$pe_slots`.

Example using the `n1ge` command:

```
$ n1ge -smp 8 a.out
```

Example using the `qsub` command:

```
qsub -cwd -j y -A "a.out" -l h_slots=1,use_mem_size=.125000,batch_sun=1 \
-pe MP 8 -N OTHERS -v N1GE_MEM=1048576 -q default \
-r y /home/admin/sun/.sgejob/OTHERS30206.sh
```

Example job script:

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
NCPUS=8
OMP_NUM_THREADS=8
export NCPUS OMP_NUM_THREADS
. /n1ge/TITECH_GRID/tools/bin_test/n1ge_bin/apps_rc/OTHERS.rc
 hoge
echo [Ending Job $JOB_ID]
```

-mpi Option

The `-mpi` option is primarily used by programs that utilize Voltaire MPI. When the `-mpi` option is specified, the `n1ge` command changes the parallel environment to PGI MPI, MPICH or HP-MPI, based on requests by the application. In the TSUBAME grid, users must specify the group if more than 32 CPUs are to be used.

Example using the `n1ge` command:

```
$ n1ge -g admin -mpi 64 a.out
```

Example using the `qsub` command:

```
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A "a.out" -P admin -l h_slots=1,use_mem_size=1,\
max_cpu_admin=1 -pe vol_mpi_8p 64 -N OTHERS -v N1GE_MEM=1048576 -q bes1 -r y \
/home/admin/sun/.sgejob/OTHERS30537.sh
```

Example job script:

```
[Job Script is ...]
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin_test/n1ge_bin/apps_rc/OTHERS.rc
MPIRUN=/usr/voltaire/mpi.pgcc.rsh/bin/mpirun_ssh
NP=$NSLOTS
PROG_ARGS="a.out"
$MPIRUN -timeout 300 -ssh_cmd /n1ge/TITECH_GRID/tools/pe/voltaire/ssh \
-np $NP -hostfile $TMPDIR/machines $PROG_ARGS
echo [Ending Job $JOB_ID]
```

The `n1ge` command uses `vol_mpi_8p` to assign eight processes per node if the number of CPUs requested with the `–mpi` option is divisible by eight. However, users can specify the number of process on each node using the following `n1ge` command. Note that 64 is the total number of CPUs desired, and 16 is the number of processes on each node.

```
$ n1ge -g admin -mpi 64:16 a.out
```

An example using the `qsub` command is below. The difference between `–mpi 64` and `–mpi 64:16` is the parallel environment used by the Sun N1 Grid Engine software. This function is also available in the `–openmpi`, `–ddi`, and `–linda` options.

```
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A "a.out"  -P admin \
-l h_slots=1,use_mem_size=1, max_cpu_admin=1 -pe vol_mpi_16p 64 \
-N OTHERS -v N1GE_MEM=1048576 -q bes1 -r y \
/home/admin/sun/.sgejob/OTHERS30756.sh
```

-ddi Option

The `–ddi` parallelization option is used in environments running the GAMESS chemistry software with DDI. More information on GAMESS and DDI can be found in “GAMESS and the Distributed Data Interface” on page 62.

Example using the `n1ge` command:

```
$ n1ge -g admin -ddi 64 rungms input.dat
```

Example using the `qsub` command:

```
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A xx -P admin \
-l h_slots=1,use_mem_size=1,max_cpu_admin=1 -pe ddi_8p 64 -N GAMESS \
-v N1GE_MEM=1048576 -q bes1 -r y \
/home/admin/sun/.sgejob/GAMESS31023.sh
```

Example job script:

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun/hama/gamess
PATH=/home/admin/sun/hama/gamess:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin_test/n1ge_bin/apps_rc/GAMESS.rc
/usr/apps/free/gamess/rungms input.dat $NSLOTS
echo [Ending Job $JOB_ID]
```

-linda Option

The -linda option is designed for use by applications using the Gaussian computational chemistry package and the TCP Linda standards for parallel programming.More information on the use of Gaussian and TCP Linda in the Sun N1 Grid Engine software environment can be found in “Gaussian with TCP Linda” on page 54.

Example using the `n1ge` command:

```
$ n1ge -g admin -linda 64 g03 test397_LindaWorker.com
```

Example using the `qsub` command:

```
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A "XX"  -P admin -l h_slots=1,Ä
use_mem_size=.029296,max_cpu_admin=1 -pe linda_8p 64 -N Gaussian -v
N1GE_MEM=2228224
-q bes1 -r y /home/admin/sun/.sgejob/Gaussian31242.sh
```

Example Job script:

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun/gau_linda
PATH=/home/admin/sun/gau_linda:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin_test/n1ge_bin/apps_rc/Gaussian.Linda.rc
if [ $RESTARTED = 1 ]; then
    echo "Checking if input file exists because this job was re-queued."
    if [ -r "test397_LindaWorker.com.$JOB_ID" ]; then
        mv test397_LindaWorker.com.$JOB_ID test397_LindaWorker.com
    else
        echo "The original input file test397_LindaWorker.com.$JOB_ID
doesn't exist."
        echo "Stopping this job for safety."
        exit 1
    fi
fi
echo "Saving original input file as test397_LindaWorker.com.$JOB_ID"
/n1ge/TITECH_GRID/tools/bin_test/n1ge_bin/apps_env/extend_files/
mod_gau_input.pl test397_LindaWorker.com
time numactl -c 6,2,3,7,5,1,0,4 -i 6,2,3,7,5,1,0,4 \
/usr/apps/isv/gaussian_linda/g03/g03 test397_LindaWorker.com
if [ -r "test397_LindaWorker.com.$JOB_ID" ]; then
    echo "Restoring original input file from
test397_LindaWorker.com.$JOB_ID"
    mv test397_LindaWorker.com.$JOB_ID test397_LindaWorker.com
fi
echo [Ending Job $JOB_ID]
```

-openmpi Option

The `-openmpi` option is designed for use with OpenMPI programs. Users can take advantage of this option to gain access to MPI2 until it is available in Voltaire MPI. The `-openmpi` option enables users to specify the parallelization and number of CPUs to use.

Example using the `n1ge` command:

```
$ n1ge -g admin -openmpi 64 a.out
```

Example using the `qsub` command:

```
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A "a.out" -P admin \
-l h_slots=1,use_mem_size=1,max_cpu_admin=1 -pe openmpi_8p 64 \
-N OTHERS -v N1GE_MEM=1048576 -q bes1 -r y \
/home/admin/sun/.sgejob/OTHERS31445.sh
```

Example job script:

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun/gau_linda
PATH=/home/admin/sun/gau_linda:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin_test/n1ge_bin/apps_rc/OTHERS.rc
MPIRUN=/usr/apps/free/openmpi1.1a2/bin/mpirun
NP=$NSLOTS
PROG_ARGS="a.out"
$MPIRUN -np $NP -machinefile $TMPDIR/machines $PROG_ARGS
echo [Ending Job $JOB_ID]
```

Other Options and Functions

While the `n1ge` command includes many options, including `–q`, `-apps`, `-mail` and more, perhaps the most important is the `-verify` debug option. This option also enables users to determine the arguments used by the `qsub` and `qrsh` commands and the job script to gain insight into how jobs are running or why they might be failing.

Example result of the `n1ge –verify` command for the AMBER job.

```
tgg075001 admin/sun> n1ge -verify -g admin -mpi 64:16 sander input.dat
*n1ge> Number of CPUs for each node => 16
*n1ge> Total number of CPUs => 64
*n1ge> Checking about admin group you specified....
*n1ge> Checking which tool have the command you specified....
*n1ge> Reading specific configuration file for each tools....
*n1ge> The version of AMBER you specified is 8.
*n1ge> Creating qsub options....
*n1ge> Submitting Job to Cluster......
[Command Line is ...]
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A
";usr;apps;isv;amber8;exe;sander+input.dat" \
-P admin -l h_slots=1,use_mem_size=1,max_cpu_admin=1 \
-pe vol_mpi_16p 64 \-N AMBER \
-v N1GE_MEM=1048576 -q bes1 -r y /home/admin/sun/.sgejob/AMBER32265.sh
[Job Script is ...]
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin_test/n1ge_bin/apps_rc/AMBER.rc
MPIRUN=/usr/voltaire/mpi.pgcc.rsh/bin/mpirun_ssh
NP=$NSLOTS
PROG_ARGS="/usr/apps/isv/amber8/exe/sander input.dat"
$MPIRUN -ssh_cmd /n1ge/TITECH_GRID/tools/pe/voltaire/ssh -np $NP \
-hostfile $TMPDIR/machines $PROG_ARGS
echo [Ending Job $JOB_ID]
```

In addition, the `n1ge` command supports version management. Users can specify a version of an application to run using an environment variable. For example, setting the environment variable `AMBER_VERSION=9` ensures the Amber9 application runs. The following output sample illustrates the use of this capability. Differences between this output and the standard output include the setting of the path for `sander` and the *.rc* file.

```
tgg075001 admin/sun> n1ge -verify -g admin -mpi 64:16 sander input.dat
*n1ge> Number of CPUs for each node => 16
*n1ge> Total number of CPUs => 64
*n1ge> Checking about admin group you specified....
*n1ge> Checking which tool have the command you specified....
*n1ge> Reading specific configuration file for each tools....
*n1ge> The version of AMBER you specified is 9.
*n1ge> Creating qsub options....
*n1ge> Submitting Job to Cluster......
[Command Line is ...]
/n1ge/bin/lx24-amd64/qsub -cwd -j y -A
";usr;apps;isv;amber9;exe;sander+input.dat" \
-P admin -l h_slots=1,use_mem_size=1,max_cpu_admin=1 \
-pe vol_mpi_16p 64 -N AMBER \
-v N1GE_MEM=1048576 -q bes1 -r y /home/admin/sun/.sgejob/AMBER32455.sh

[Job Script is ...]
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin_test/n1ge_bin/apps_rc/AMBER.9.rc
MPIRUN=/usr/voltaire/mpi.pgcc.rsh/bin/mpirun_ssh
NP=$NSLOTS
PROG_ARGS="/usr/apps/isv/amber9/exe/sander input.dat"
$MPIRUN -ssh_cmd /n1ge/TITECH_GRID/tools/pe/voltaire/ssh -np $NP \
-hostfile $TMPDIR/machines $PROG_ARGS
echo [Ending Job $JOB_ID]
```

Utility Commands for Users

This section describes other utility commands available to users.

The qjobs Command

The `qstat` command cannot display information on finished jobs, and the format of `qacct` command often is too difficult for users to decipher. To help users understand job statistics, the `qjobs` command displays both the output of the `qstat` command and finished jobs. The output below lists the options available for the `qjobs` command.

```
tgg075001 admin/sun> qjobs -help
usage: qjobs [options]
  [-help]                          :: display this message.
  [-a [hours] [-rq]]               :: show PEND RUN FINISHED JOBS.
                                      -rq show RE-QUEUED JOBS.
  [-r]                             :: show only RUN JOBS.
  [-p]                             :: show only PEND JOBS.
  [-f [hours] [-rq]]               :: show only FINISHED JOBS.
                                      -rq show RE-QUEUED JOBS.
  [-u [user_list | all]            :: show only jobs of this user.
                                   "-u all" show the jobs of all user.
  --
  user_list                        :: username[,username ...]
```

Internally, the `qjobs` command uses the `qstat` command for `PEND` and `RUN` jobs, and uses accounting files in the *$SGE_ROOT/$SGE_CELL/common* directory for `FINISHED` jobs. The example below displays the results of the `qjobs –f 200` command, where 200 specifies all jobs that finished within 200 hours be shown. The `slots` value indicates the number of CPUs used by the job, `cpu` is the CPU time used in seconds, and memory is the maximum size of virtual memory in MB used by the job.

```
tgg075003 admin/sun> qjobs -f 200
[ Finished Jobs (200 hours ago -> 10/25/2006 08:01:16)]
job-ID  type  name      user  state submit time          submit host executed queue      slots cpu       memory
------------------------------------------------------------------------------------------------------------------
347254  BACK  OTHERS    sun   DONE  11/02/2006 16:10:19  tgg075003   supercon@tgg072185  8     0.0       61.53
347254  BACK  OTHERS    sun   -     -                    tgg075003   supercon@tgg072185  8     485.0     18105.73
347239  FORE  MATLAB    sun   DONE  11/02/2006 15:59:50  tgg075003   high@tgg075007      1     1.4       1597.54
344753  FORE  LOGIN     sun   DONE  11/01/2006 21:19:36  login1      interact@tgg075003  1     0.2       44.30
344297  BACK  OTHERS    sun   DONE  11/01/2006 15:51:45  tgg075003   supercon@tgg072188  16    0.0       91.54
344297  BACK  OTHERS    sun   -     -                    tgg075003   supercon@tgg072187  16    964.0     9981.16
344297  BACK  OTHERS    sun   -     -                    tgg075003   supercon@tgg072188  16    964.0     9981.16
311853  BACK  Gaussian  sun   EXIT  10/23/2006 20:07:21  tgg075003   A@tgg074027         32    1062348.0 8470.72
311853  BACK  Gaussian  sun   -     -                    tgg075003   A@tgg074013         32    407293.6  8359.15
311853  BACK  Gaussian  sun   -     -                    tgg075003   A@tgg074011         32    406940.0  8359.15
311853  BACK  Gaussian  sun   -     -                    tgg075003   A@tgg074024         32    410804.0  8359.15
```

The qcomplexes Command

The `qcomplexes` command displays the use of each FLEXlm `FEATURE`. In the example output below, note that `NJOBS` is the number of jobs which use or will use the `FEATURE`, and `CURRENT` is the available number of `FEATURE`s obtained by `load_sensor`.

```
sgeadmin@tggn1ge1:~> qcomplex -v
*qcomplex> Getting feature information in this cluster....
*qcomplex> Checking for running jobs....
*qcomplex> Checking for pending jobs....
*qcomplex> Checking for suspending jobs....

-----------------------------------------------------------------------
COMPLEX_NAME            STATUS  NJOBS   PEND    RUN     SUSP    CURRENT
-----------------------------------------------------------------------
.....
CAMPUS                  OK      0       0       0       0       2846
MATLAB                  OK      0       0       0       0       5
MSI_TokenR              OK      0       0       0       0       28
Robust_Toolbox          OK      0       0       0       0       10
SIMULINK                OK      0       0       0       0       9
Signal_Toolbox          OK      0       0       0       0       10
Simulink_Control_Design OK      0       0       0       0       10
abaqus                  OK      7       3       4       0       42
aqua                    OK      0       0       0       0       80
cae                     OK      3       0       3       0       2
.....
```

The qbqueues Command

Similar to the `qstat –g c` command, the `qbqueues` command displays the number of jobs on each queue. In the following example output, 5387(1199) indicates 1199 jobs using a total of 5387 CPUs.

```
sgeadmin@tggn1ge1:~> qbqueues
*qbqueues> Getting queue information in this cluster....
*qbqueues> Getting information about running jobs....
*qbqueues> Getting information about pending jobs....
*qbqueues> Summarizing those information about queues and jobs....
QUEUE_NAME   STATUS              MAX         NJOBS          PEND            RUN           SUSP
--------------------------------------------------------------------------------------------
A            Open:Active        1920      893( 147)      128(   4)      765( 143)        0(    0)
interactive  Open:Active        4000      151( 151)        0(   0)      151( 151)        0(    0)
bes1         Open:Active        1920     1172( 511)      192(   1)      980( 510)        0(    0)
default      Open:Active         624     1126( 743)      546( 518)      580( 225)        0(    0)
avs          Open:Active         320        0(   0)        0(   0)        0(   0)        0(    0)
B            Open:Active        1920     1392(  56)       48(   1)     1344(  55)        0(    0)
sla1         Open:Active        1920     1584(  10)      144(   1)     1440(   9)        0(    0)
supercon     Open:Active         704        0(   0)        0(   0)        0(   0)        0(    0)
sas          Open:Active          16        0(   0)        0(   0)        0(   0)        0(    0)
all.q        Open:Active       10496        0(   0)        0(   0)        0(   0)        0(    0)
high         Open:Active         192      155( 111)       36(   7)      119( 104)        0(    0)
mopac        Open:Active          48        8(   2)        0(   0)        8(   2)        0(    0)
--
TOTAL        -                 24080     6486(1736)     1099( 537)     5387(1199)        0(    0)
```

Chapter 6
Resource Management

This chapter describes the resource management aspects of the TSUBAME grid.

Integration of FLEXlm and Sun N1 Grid Engine Software

In the TSUBAME grid, FLEXlm is the networked license server. The Sun N1 Grid Engine software integrates with FLEXlm in order to:

- Prevent the failure of obtaining software licenses after starting jobs on execution nodes.
- Enable the Sun N1 Grid Engine software to re-queue jobs if it knows a license cannot be obtained.

Figure 6-1 illustrates the mechanism by which the Sun N1 Grid Engine software obtains license information from the FLEXlm license server.

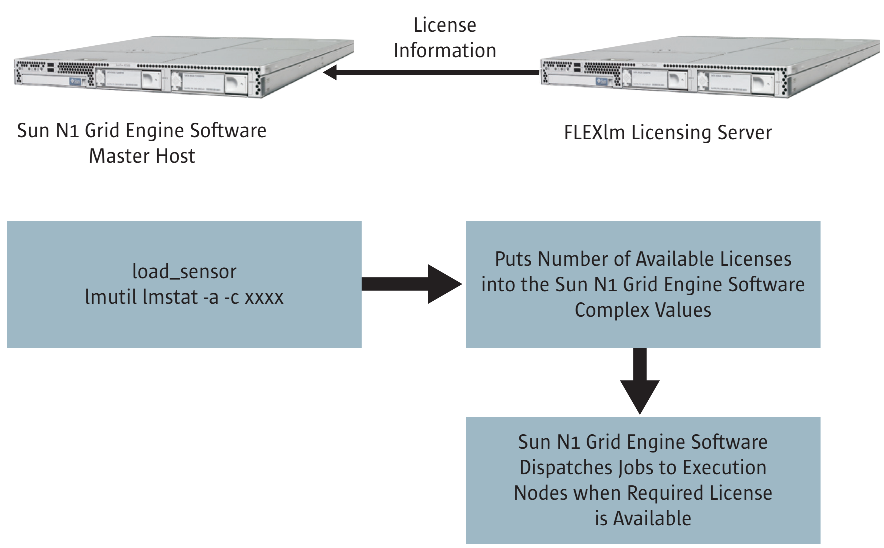

Figure 6-1. The license access mechanism in the TSUBAME grid

The Sun N1 Grid Engine software includes a `load_sensor` parameter to which a script or binary can be assigned. In the TSUBABME grid, the Sun N1 Grid Engine daemon, sge_execd, runs a script in this manner regularly on the execution nodes. The default execution interval is 40 seconds. To obtain information from a license server, simply assign the FLEXlm command to the `load_sensor` parameter, such as `lmutil lmstat –a –c` *xxxx*. Once the output of the `lmutil` command is adapted, the Sun N1 Grid Engine software can take the returned information and place it into its internal variables, or complexes. This technique enables the Sun N1 Grid Engine software to manage license aware jobs and minimize the likelihood of being unable to obtain a license for an application running on execution nodes. Note it is possible for a license failure to occur, as the interval between command runs is 40 seconds. The re-queue mechanism of the Sun N1 Grid Engine software can help in this case by resubmitting jobs.

When a user submits a job which requires a software license, the Sun N1 Grid Engine software checks to see if a license is available using the licensing information obtained by the `load_sensor` function. If a license is available, the Sun N1 Grid Engine software dispatches the job to the execution nodes (Figure 6-2). The software checks license availability once again on the execution node prior to starting the job using the `lmutil` command. If a license is not available, the Sun N1 Grid Engine software re-queues the job as a pending job. These functions eliminate the need for users to keep tracking of software licenses.

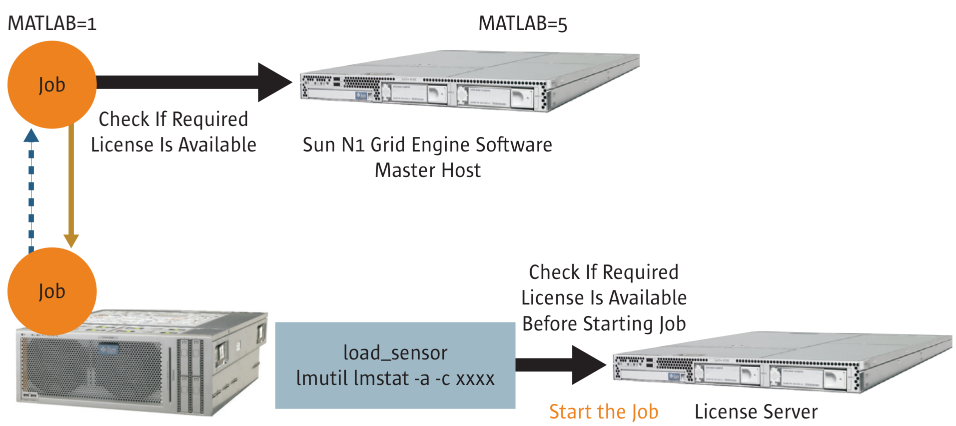

Figure 5-2. The TSUBAME re-queues jobs if licenses cannot be obtained

Obtaining License Information from FLEXlm

The following steps outline the procedure used to enable the Sun N1 Grid Engine software to obtain licensing information from the FLEXlm software.

1. Write a script which obtains licensing information from the FLEXlm software and format output as follows:

```
global:feature_name:the_available_number
```

2. Create global complexes for each feature.
3. Configure the `load_sensor` function in the master and shadow master hosts.

```
qconf -mconf hostname
```

4. Submit jobs requiring licenses by using complexes.

Writing the Script

The sample `get_feature_info.pl` below outputs the information from the `lmutil -a -c` *xxxx* command in the form `global:feature_name:available_number`. The important entries to note are those related to users and licenses, such as *Users of MATLAB: (Total of 10 licenses issued; Total of 4 licenses in use)*.

```
.....
Users of CAMPUS:  (Total of 3000 licenses issued;  Total of 231 licenses in use)

  "CAMPUS" v2007.0331, vendor: MSC
  floating license

    XXXXX tgg075003 /dev/ttyrc CAMPUS:PATRAN (v2005.0701) (tggls/27000 767), start Tue 10/31 19:55, 77 licenses
    XXXXX tgg075004 /dev/pts/4 CAMPUS:PATRAN (v2005.0701) (tggls/27000 260), start Tue 10/31 20:08, 77 licenses
    XXXXX tgg075004 /dev/ttyr9 CAMPUS:PATRAN (v2005.0701) (tggls/27000 562), start Tue 10/31 20:10, 77 licenses
.....
Users of MATLAB:  (Total of 10 licenses issued;  Total of 4 licenses in use)

  "MATLAB" v15, vendor: MLM
  nodelocked license, locked to "ID=313047"

    XXXXX tgg075002 /dev/ttyq4 (v15) (tggls/27000 756), start Mon 10/30 16:07
    XXXXX tgg075003 /dev/ttyrb (v15) (tggls/27000 1152), start Tue 10/31 18:49
    XXXXX tgg075004 /dev/ttyrd (v15) (tggls/27000 1284), start Tue 10/31 17:58
    XXXXX tgg075004 /dev/ttyp3 (v15) (tggls/27000 610), start Thu 10/12 23:06
.....
```

In this example, six MATLAB licenses are available. As a result, the script outputs the following information:

```
global:MATLAB:6
```

Any scripting language can be used, including Perl, shell and Ruby, as well as programming languages such as C and Java. Scripts must simply:

- Obtain license information using the `lmutil lmstat –a –c XXX` command
- Output license information in the form `global:feature_name:available_number`

Sample output from the `get_feature_info.pl` script is listed below.

```
sgeadmin@tggn1ge1:/n1ge/TITECH_GRID/tools/util> ./get_feature_info.pl
global:Ludi_Score_TokenR:999
global:pgdbg:25
global:standard:62
global:MS_castep_ui_TokenR:999
global:ProteinFamilies_Client_TokenR:999
global:CHARMM_TokenR:999
global:MS_compass_TokenR:993
global:CMATH:10000
global:IMSLFPC:10000
global:MS_powdersolve_TokenR:999
global:pghpf-linux86:25
global:pgprof:25
global:Simulink_Control_Design:10
global:PATRAN:1
global:abaqus:52
global:health_TokenR:999
global:Ludi_TokenR:999
global:MATLAB:6
.....
```

Creating Complexes for Features

The Sun N1 Grid Engine software stores the available number of licenses automatically in internal variables if load_sensor outputs the information as described earlier. However, it is necessary to create complexes for the features output by `load_sensor` using a feature name. Table 6-1 details then entry for the MATLAB application. Be sure to create complexes for each feature output by the `load_sensor` function using these guidelines.

Table 6-1. Complex description for the MATLAB application

Name	Shortcut	Type	relop	requestable	consumable	default	urgency
MATLAB	MATLAB	INT	<=	YES	NO	0	0

Configuring Load Sensor in the Master Host and Shadow Master Host

The next step is to write the script that is assigned to the `load_sensor` function. A sample `load_sensor` script is listed below. By setting the script as the `load_sensor` on both the master host and shadow master host, `load_sensor` is able to continue to retrieve license information from the license server in the event of a failure of the master host. More detailed information on load sensor scripts can be found in the Sun N1 Grid Engine documentation available at *docs.sun.com*

```
#!/bin/sh
GET_FEATURE_INFO="XXXX/get_feature_info.pl"
while [ 1 ]; do
    # wait for input
    read input
    result=$?
    if [ $result != 0 ]; then
        exit 1
    fi
    if [ "$input" = "quit" ]; then
        exit 0
    fi
    if [ -f $SGE_ROOT/$SGE_CELL/common/act_qmaster ]; then
        ACT_MASTER=`cat $SGE_ROOT/$SGE_CELL/common/act_qmaster`
        if [ "$ACT_MASTER" = "" ]; then
            ERROR_STATUS="TRUE"
        fi
    else
        ERROR_STATUS="TRUE"
    fi
    if [ "$HOSTNAME" = "$ACT_MASTER" ]; then
        if [ "$ERROR_STATUS" = "FALSE" ]; then
            echo "begin"
            `$GET_FEATURE_INFO`
            echo "end"
        fi
    fi
done
exit 0
```

Confirm the Sun N1 Grid Engine software can determine the number of available licenses by executing the qhost command. Ensure the results displayed are similar to the output below.

```
$ qhost -F -h tggn1ge1 |grep gl:|sort|uniq
   gl:Biopolymer_TokenR=999.000000
   gl:CAMPUS=2923.000000
   gl:CHARMM_TokenR=999.000000
   gl:CMATH=10000.000000
   gl:CSTAT=10000.000000
   gl:Control_Toolbox=10.000000
   gl:DS_Analysis_TokenR=999.000000
   gl:DS_ModelingVisualizer_TokenR=999.000000
   gl:DS_ProteinFamilies_TokenR=999.000000
   gl:DS_ProteinSimSrch_TokenR=999.000000
   gl:DelPhi_TokenR=1998.000000
   gl:IMSLFPC=10000.000000
   gl:LIGANDFIT_TokenR=999.000000
   gl:LIGSCORE_TokenR=999.000000
   gl:License_Holder=996.000000
   gl:Ludi_Genfra_TokenR=999.000000
   gl:Ludi_Score_TokenR=999.000000
   gl:Ludi_TokenR=999.000000
   gl:MATLAB=6.000000
.....
```

Submitting Jobs Using Complexes

Users must submit jobs using the –l feature_name=XX option to the qsub or qrsh command to ensure the Sun N1 Grid Engine software can manage license-aware jobs by using the license information obtained by load_sensor. In the TSUBAME grid, the n1ge command automatically utilizes the –l option if the application to be run needs a networked license. In addition, users must know how many licenses each application requires. For example, the MATLAB application requires one MATLAB feature, while the PATRAN application requires at least 77 CAMPUS features. As a result, users submitting MATLAB or PATRAN jobs in the TSUBAME grid must specify the following options. Because keeping tracking of license requirements can be complicated, the n1ge wrapper script eases the management of the grid.

```
$ qrsh –l MATLAB=1 .... matlab
$ qrsh –l CAMPUS=77 .... patran
```

Implementing the Re-Queue Mechanism

Implementing the job re-queue mechanism involves writing a script to obtain license information from the license server, confirming the availability of the license, and submitting and testing the job.

Writing the Script

Queue configuration includes the queue_prolog parameter, which executes before a job runs to confirm the availability of the license. This step helps minimize the likelihood a license cannot be obtained on the execution nodes. The script assigned to the queue_prolog parameter must:

- Determine the required feature name of the job through an environment variable
- Obtain license information from the license server
- Confirm whether the required feature is available, and return an exit code of 99 if the feature is not available

The easiest way to pass the environment variable to queue_prolog is to use the –v option of the qsub or qrsh command. The following examples illustrate the options to specify when submitting a job which requires MATLAB=1, as well as PATRAN. Values such as FEATURE=MATLAB=1 and FEATURE=CAMPUS=77 export to the execution nodes, enabling queue_prolog to use these values as environment variables.

```
$ qrsh –l MATLAB=1 –v FEATURE=MATLAB=1 .... matlab
$ qrsh –l CAMPUS=77 –v FEATURE=CAMPUS=77 .... patran
```

The Sun N1 Grid Engine software can re-queue background jobs submitted by the `qsub` command. If a job terminates with an exit code of 99, the Sun N1 Grid Engine software automatically re-queues the job. As a result, a routine must be created to handle the case where the job requires features that are not available. In this case, terminating with an exit code of 99 enables the job to be re-queued.

A sample `queue_prolog` script follows. This `check_license.pl` script is nearly identical to the `get_feature_info.pl` script discussed earlier in this document. Differences include confirming if a granted feature is available, and if the required number is available, specifying GO as the output.

```
#!/bin/sh
.....
CHECK_LICENSE="$SGE_ROOT/$SGE_CELL/tools/util/check_license.pl"
if [ ! -x $CHECK_LICENSE ]; then
logging_error Can not execute $CHECK_LICENSE
fi

if [ "$JOB_SCRIPT" = "QRSH" ] || [ "$JOB_SCRIPT" = "INTERACTIVE" ]; then
TYPE="FORE"
else
    TYPE="BACK"
fi
..........
############################################################
# Checking Licenses
#
if [ "$FEATURE" != "" ]&&[ "$TYPE" = "BACK" ]; then
check_num_feature=`echo $FEATURE|grep ":"`
    if [ "$check_num_feature" != "" ]; then
        TARGET_FEATURE=`echo $FEATURE|sed -e 's/:/ /g'`
    else
        TARGET_FEATURE=$FEATURE
    fi
    for i in $TARGET_FEATURE
    do
        if [ `$CHECK_LICENSE $i` != "GO" ]; then
            exit 99
        fi
done
fi

exit 0
```

Submitting Jobs and Testing

The `qsub` command must be used to re-queue jobs. Be sure to confirm the re-queue mechanism is functional.

Configuring the Operating System

Each execution node in the TSUBAME grid denies login attempts that do not use the Sun N1 Grid Engine software by executing the */etc/nologin* script. In addition, execution nodes do not allow the running of cron jobs by general users. If users set cron jobs to run, execution nodes can be monopolized by users. As a result, it is important to deny users the ability to run cron jobs by configuring the */var/spool/cron/allow* file.

Setting Job Limits

Tokyo Tech needed to be able to set a limit on the number of CPUs a member of a group can use at any given time. As a result, the TSUBAME supercomputer grid uses a job limit per user (JL/U) for the `default` and `high` queues, and a job limit per group (JL/G) for the `bes1` and `bes2` queues. While the Sun N1 Grid Engine software has the ability to set a job limit per user, it applies to the entire software environment rather than a specific queue.

Before explaining how to limit the number of CPUs a user can access, it is important to discuss the behavior of consumable complexes in the Sun N1 Grid Engine software. Consumable complexes are defined with `consumable=YES`. For example, `slots` is a consumable complex. If it is necessary to decrease the value of a complex once a job begins execution, the complex is a consumable complex. Be sure to set `consumable=YES` for any complexes with this characteristic. For example, `h_slots` must be a consumable complex because it refers to the available number of CPUs on an execution node — a value that changes over time. As a result, the complexes that define the maximum number of CPUs that can be access by a user or group must be consumable complexes.

The key concern is to express the number of CPUs a job should use. For example, specify the following option to the `qsub` command when a job should use one CPU. The `h_slots` value is set to 1 to represent one CPU.

```
$ qsub –l h_slots=1 test.sh
```

Use the following `qsub` command to specify a job should use eight CPUs.

```
$ qsub –l h_slots=1 –pe MP 8 test.sh
```

The parallel environment must be specified using the `-pe` option of the `qsub` or `qrsh` command one more then two CPUs are to be used. As a result, the value of `h_slots` becomes `8` because the required resources specified by the `–l` option are multiplied by the value of `–pe PE_NAME`. While `-l h_slots=8` indicates eight CPUs should be used, the result of the `qstat` command varies in this case. For example, `-l h_slots=8` uses only one slot as shown below.

```
job-ID   prior     name     user   state   submit/start at       queue          slots   ja-task-ID
-------------------------------------------------------------------------------------------------
343237   0.50500   test.sh  sun    r       10/31/2006 23:32:25   default@tgg    1       075033
```

As another example, `–l h_slots=1 –pe MP 8` uses eight slots to obtain access to eight CPUs:

```
job-ID   prior     name     user   state   submit/start at       queue          slots   a-task-ID
-------------------------------------------------------------------------------------------------
343236   0.50637   test.sh  sun    r       10/31/2006 23:31:10   default@tgg    8       075033
```

The following options are specified automatically by the `n1ge` command when the user `sun` submits a job to the `default` or `high` queue.

```
$ qsub –l h_slots=1,batch_sun=1 –pe MP 8 xxxx
```

The following options are specified automatically by the `n1ge` command when the user `sun` submits a job to the `bes1` or `bes2` queues, which are limited by group. The `max_cpu_admin=1` option is used to limit access by the `admin` group to which the user `sun` belongs.

```
$ qsub –l h_slots=1,max_cpu_admin=1 –pe MP 8 xxxx
```

It is possible to set the job limit per user and the job limit per group using the same complexes. Simply set `batch_sun` and `max_cpu_admin` to one divided by the number of CPUs required by the job. This calculation ensure the value of `batch_sun` and `max_cpu_admin` ultimately becomes one job. The following example shows the options to specify for a job requiring eight CPUs. Note that 0.125 * 8 = 1.00

```
$ qsub –l h_slots=1,batch_sun=0.125 –pe MP 8 xxxx
```

An example for a job requiring 16 CPUs is displayed below.

```
$ qsub –l h_slots=1,batch_sun=0.0625 –pe MP 16 xxxx
```

If the relation between `-l` and `-pe` is know, it is possible to realize job limits on users and groups to set the maximum number of CPUs a user can utilize at the same time in the Sun N1 Grid Engine environment.

Setting the Maximum Memory Size for a Job

It is possible for the Linux operating system kernel to hang if a user uses more memory (virtual memory plus swap) than is available on the system. This is an important consideration in an environment which uses fat SMP machines as execution nodes to run a large number of jobs. This issue can have a cascading effect on jobs in the system, causing all jobs to fail due to the actions of one job. To address these concerns, the TSUBAME grid takes the following actions.

- *Specifies the memory size for jobs when users submit jobs*
  The Sun N1 Grid Engine scheduler prevents the use of memory than is available on the system by using the `use_mem_size` parameter. The following options are specified by the `n1ge` command automatically when users submit jobs.

  Example: `n1ge` command:

  ```
  $ n1ge –mem 4 xxxx
  ```

  Example `qsub` command:

  ```
  qsub –l h_slots-1,use_mem_size=4
  ```

- *Limits the memory size of a job to the size specified by the user with ulimit –v*
  Using the example above, the Sun N1 Grid Engine scheduler allocates an execution node with 4 GB of free memory to a job by using `use_mem_size` complexes. This requires the actual memory used by the job to be less than or equal to 4 GB. If the job uses more than 4 GB of memory, then there is no point in specifying a memory size upon job submittal. Therefore, the TSUBAME grid kills jobs that use more memory than is specified by users.

  While the Sun N1 Grid Engine software includes a function that limits the memory size for a queue — all jobs that run in the queue are limited to the specified amount of memory. This function can be used if all execution nodes incorporate no more than two CPUs. To limit memory usage for each job individually, execute `ulimit –v` for every job. This applies to single jobs as well as parallel jobs like MPI, TCP Linda, and DDI. This function can be implemented easily by adding `ulimit -v` in the job script created by the `n1ge` command. To keep users from modifying the setting, the TSUBAME grid implements this functionality in the `starter_method` for each queue.

Single Jobs

The output below details the queue configuration for the TSUBAME grid. The `starter_method` modifications are shown in bold type.

```
qname                 bes1
.....
prolog                /n1ge/TITECH_GRID/tools/util/queue_prolog
epilog                /n1ge/TITECH_GRID/tools/util/queue_epilog
shell_start_mode      posix_compliant
starter_method        /n1ge/TITECH_GRID/tools/util/job_starter
suspend_method        /n1ge/TITECH_GRID/tools/util/suspend_method
resume_method         /n1ge/TITECH_GRID/tools/util/resume_method
```

The `job_starter` Perl script monitors the standard output and error messages of jobs. If errors occur, `job_starter` exits with status code `99` and the Sun N1 Grid Engine software re-queues the job automatically. However, if `job_starter` understands the error messages output when the application fails to obtain a license, `job_starter` re-queues the job automatically. The logic for this functionality can be found in the `if($num_error_messages !=0)` clause, and `$num_error_messages` is the number of error messages output by the application. In addition, `job_starter` executes `ulimit –v xx` in order to limit the memory size of job. A portion of the `job_starter` script is listed below. Note the entries shown in bold type.

```
if (("$JOB_SCRIPT" eq "QRSH") || ("$JOB_SCRIPT" eq "INTERACTIVE")) {
my $EXIT_CODE=system("$N1GE_EXEC @ARGV") / 256;
    exit ($EXIT_CODE);
} else {
    if ($num_error_messages != 0) {
        open OUTPUT, "$N1GE_EXEC @ARGV 2>&1|";
        while (<OUTPUT>) {
            chomp;
            $STD_MESSAGES = $_;
            foreach(@error_messages) {
                if ($STD_MESSAGES =~ /($_)/ ) {
                    print "$STD_MESSAGES \n";
                    print "Rescheduling this JOB....\n";
                    exit (99);
                }
            }
            if ($STD_MESSAGES =~ /EXIT_CODE:/) {
                $_ =~ s/EXIT_CODE://;
                exit $_;
            }
            print "$STD_MESSAGES \n";
            flush(STDOUT);
        }
} else {
        my $EXIT_CODE=system("$N1GE_EXEC @ARGV") / 256;
        exit ($EXIT_CODE);
    }
}
```

The `job_starter` script executes the `n1ge_exec` shell script to run the job. Of course, `n1ge_exec` executes the `ulimit` command. A portion of the `n1ge_exec` script is listed below. The key point to notice is the `ulimit –v $N1GE_MEM` command. The job sent as an argument to `job_starter` when `job_starter` is setup in the queue configuration. In effect, `@ARGV` in `job_starter` and `$@` in `n1ge_exec` handle the actual job. In the `n1ge_exec` script, memory size is limited by executing the `ulimit` command prior to the `$@` command. All that remains is to pass the value of the `–mem` option of the `n1ge` command to the job as the `$N1GE_MEM` environment variable. This can be accomplished using the –v option of the qsub or qrsh command, as described in “Implementing the Re-Queue Mechanism” on page 35.

```
.....
if [ "$JOB_SCRIPT" = "QRSH" ] || [ "$JOB_SCRIPT" = "INTERACTIVE" ]; then
TYPE="FORE"
else
TYPE="BACK"
fi
.....
exit_job() {
EXIT_CODE=$1
    if [ "$TYPE" = "BACK" ]; then
        echo "EXIT_CODE:$EXIT_CODE"
    fi
    exit $EXIT_CODE
}
.....
ulimit -v $N1GE_MEM
if [ "$NSLOTS" = "1" ]; then
$@
    result=$?
    exit_job $result
else
    .....
$@
    result=$?
    exit_job $result
fi
```

The `n1ge` command converts `–mem 4` into `-l use_mem_size=4 –v N1GE_MEM=4456448` automatically for use as an option to the `qsub` and `qrsh` commands. Note that 4456448 = 4 * 1024 * 1024 + buffer, in KB. The `n1ge_exec` command executes `ulimit –v 4456448` by using the `$N1GE_MEM` environment variable on the queue. As a result, the job will be killed if it uses more memory than is specified by the `–mem` option of `n1ge` command.

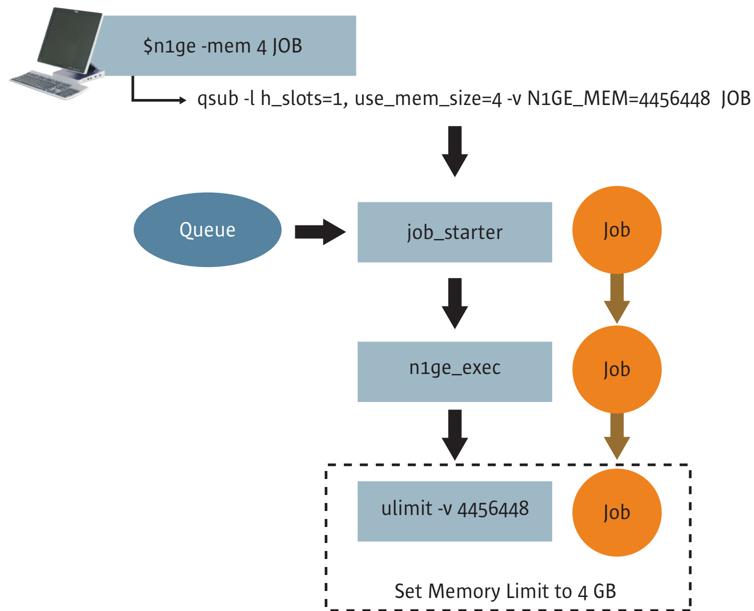

Figure 6-3. Setting the limit of the memory that can be used

Symmetric Multiprocessing Jobs

For jobs which use threads, such as OpenMP, `use_mem_size` should not be set to the value specified by the `–mem` option of the `n1ge` command. Figure 6-4 illustrates the flow used to limit the memory size of a job which uses multiple CPUs on an execution node.

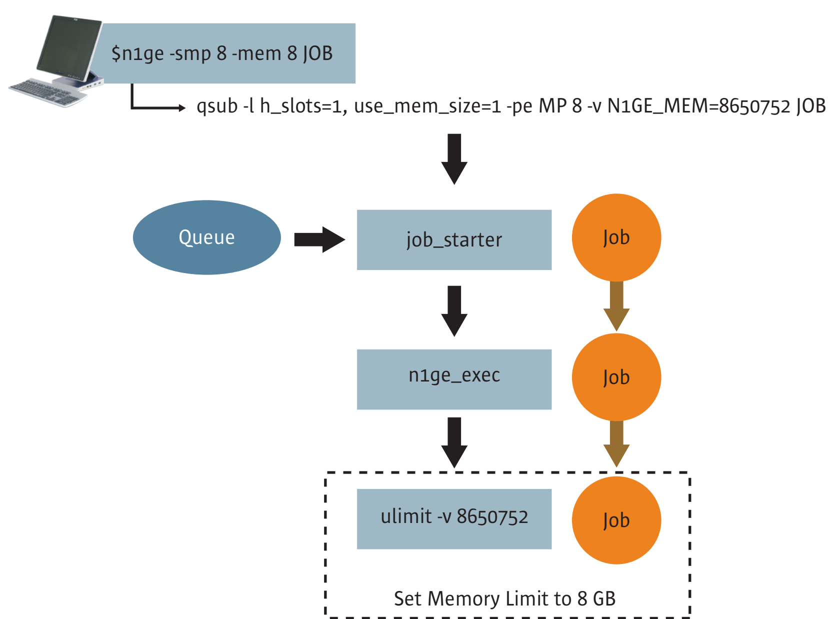

Figure 6-4. Setting the memory limit for threaded jobs

While it is important to specify the parallel environment for jobs requiring multiple CPUs, the value of `use_mem_size` is multiplied by the number of CPUs. In the example above, the Sun N1 Grid Engine software is able to determine that the job requires 8*1 = 8 GB of memory. The right value of `use_mem_size` must be one because the user requires 8 GB of memory for the job. However, the value of the `$N1GE_MEM` environment variable must be set to 8 GB, otherwise the jobs requiring 8 GB of memory will only be able to use 1 GB of memory.

Parallel Jobs

Care must be taken when limiting the size of memory for MPI jobs which run on several nodes. Figure 6-5 illustrates the flow to limit the memory size for a job which uses Voltaire MPI. The first portion of the flow is similar to the flow for single and SMP jobs.

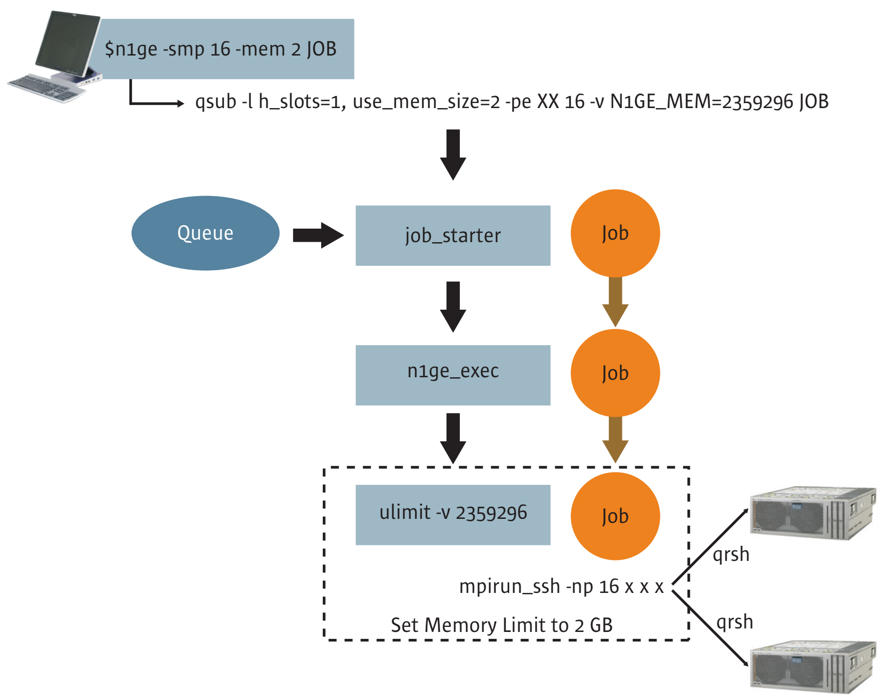

Figure 6-5. Setting the memory limit for parallel jobs

The issue is that `ulimit –v` works only for the `mpirun_ssh` command. For Voltaire MPI, `n1ge_exec` can limit the memory size for the following commands.

```
sgeadmin 13376     1 13376 sge_execd
sgeadmin 13455 13376 13455  \_ sge_shepherd
sun      13570 13455 13570  |   \_ job_starter
sun      13571 13570 13570  |       \_ n1ge_exec
sun      13579 13571 13570  |           \_ 344298
sun      13627 13579 13570  |               \_ mpirun_ssh
sun      13628 13627 13570  |                   \_ qrsh
sun      13647 13628 13570  |                   |   \_ ssh
sun      13629 13627 13570  |                   \_ qrsh
sun      13646 13629 13570  |                       \_ ssh
```

Note that `344298` is a job script. However, only the `mpirun_ssh`, `qrsh`, and `ssh` processes are executed by `mpirun_ssh`. In addition, `n1ge_exec` is unable to limit the size of memory for processes because some MPI programs are executed on other nodes using the `ssh` command. By modifying the `mpirund` command, each rank can be managed using the script shown below. Be sure to save the original `mpirund` file.

```
#!/bin/sh
#
# mpirund
#
# [Overview]
#       This script is wrapper script for mpirund in order to
#       limit the maximum virtual memory size of each mpi processes.
#       The original mpirund binary is re-named to mpirund.orig.
#       The N1GE_MEM environment variable is exported by N1 Grid Engine.
#
#       version 0.1
#               First creation
#               by Sun Microsystems.K.K. 4/18/2006
#
N1GE_MEM=${N1GE_MEM:="unset"}
echo "[mpirund@$HOSTNAME] The maximum size of Virtual Memory =>
$N1GE_MEM kb."
if [ "$N1GE_MEM" != "unset" ]; then
    ulimit -v $N1GE_MEM
else
    echo "[mpirund@$HOSTNAME] Can't find N1GE_MEM env variable."
fi
/usr/voltaire/mpi.pgcc.rsh/bin/mpirund.orig $@
```

The `$N1GE_MEM` environment variable can be used in the shell script, as environment variables are passed to all ranks in Voltaire MPI when `mpirun_ssh` is executed. The memory size of each MPI program executed by `mpirund.orig` can be limited if `ulimit –v` can be executed in `mpirund`.

Voltaire MPI. The first portion of the flow is similar to the flow for single and SMP jobs.

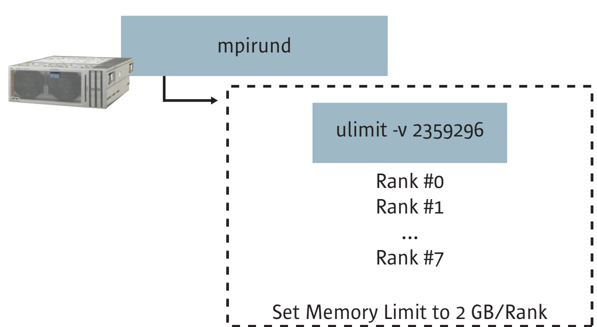

Figure 6-6. Setting the memory limit for MPI jobs

The process tree for execution nodes in Voltaire MPI follows. The memory size of *a.out* can be limited because it runs under the customized `mpirund` command.

```
sgeadmin 13376     1 13376 sge_execd
sgeadmin 13644 13376 13644  \_ sge_shepherd
root     13645 13644 13645      \_ sshd
sun      13653 13645 13645          \_ sshd
sun      13654 13653 13654              \_ qrsh_starter
sun      13749 13654 13749                  \_ job_starter
sun      13750 13749 13749                      \_ sh
sun      13759 13750 13749                          \_ mpirund
sun      13760 13759 13749                              \_ mpirund.orig
sun      13761 13760 13749                                  \_ a.out
sun      13762 13760 13749                                  \_ a.out
sun      13763 13760 13749                                  \_ a.out
sun      13764 13760 13749                                  \_ a.out
sun      13765 13760 13749                                  \_ a.out
sun      13766 13760 13749                                  \_ a.out
sun      13767 13760 13749                                  \_ a.out
sun      13768 13760 13749                                  \_ a.out
```

Chapter 7

Integrating Applications with Sun N1 Grid Engine Software

Integrating applications, such as Amber, Gaussian with TCP Linda and MATLAB, with the Sun N1 Grid Engine software focuses on the following three tasks:

- *Creating the wrapper script for the qrsh and qsub commands*
  Integrating applications and the Sun N1 Grid Engine software is key to making the grid as transparent to users as possible. The wrapper script helps this effort. For the integration to be smooth, it is important to understand the types of applications users wants to run, prepare the application configuration using *.rc* files and Sun N1 Grid Engine software functions, and write a job script for the `qsub` or `qrsh` command.
- *Creating ssh for the Sun N1 Grid Engine software*
  Creating `ssh` for the Sun N1 Grid Engine software is essential for the proper execution of applications that incorporate a shell environment or use MPI. More information can be found in “SSH for Sun N1 Grid Engine Software” on page 13.
- *Configuring the parallel environment*
  Many applications take advantage of parallelization mechanisms, such as MPI, DDI, Linda, and others. As a result, integration of the parallel environment with the Sun N1 Grid Engine software is key to successful application execution and optimal performance.

Simple Jobs

Simple jobs, such as user scripts, commands such as `id`, and applications like MATLAB, can be integrated in a straightforward manner with the Sun N1 Grid Engine software.

id Utility

When the `id` command is executed via the `n1ge` command, the following options to the `qsub` and `qrsh` commands and the job script are created automatically.

Example `n1ge` command:

```
$ n1ge id
```

Example options to the `qsub` command:

```
qsub -cwd -j y -A "id" -l h_slots=1,use_mem_size=1.000000,batch_sun=1 \
-N OTHERS -v N1GE_MEM=1048576 -q default \
-r y /home/admin/sun/.sgejob/OTHERS10382.sh
```

Example job script:

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/OTHERS.rc
 id
echo [Ending Job $JOB_ID]
```

While the `qsub` command is not present in the job script, the job script does need to read the *apps.sh* file which sets environment variables for applications, including `PATH`, `LD_LIBRARY_PATH`, `LM_LICENSE_FILE`, and more.

MATLAB Application

Options to the `qsub` and `qrsh` commands, as well as an example job script, are listed below for the MATLAB application running via the `n1ge` command.

Example `n1ge` command:

```
$ n1ge matlab
```

Example options to the `qrsh` command:

```
qrsh -cwd -A ";us;home;admin;sun;apps;isv;matlab72;bin;matlab" \
-l h_slots=1,use_mem_size=2.000000,batch_sun=1,MATLAB=1 -N MATLAB -v FEATURE=MATLAB=1, \
N1GE_MEM=2359296 -q high,default  /home/admin/sun/.sgejob/MATLAB11219.sh
```

Example job script:

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
/usr/apps/isv/matlab72/bin/matlab
echo [Ending Job $JOB_ID]
```

The `n1ge` command notices the application to be run is MATLAB based on the arguments to the `n1ge` command. As a result the `n1ge` command automatically specifies the following options to the `qrsh` command. Note that MATLAB requires at least 2 GB of memory to run. Furthermore, the `n1ge` command runs MATLAB using the `qrsh` command because MATLAB is an interactive application, and sets the application path to */usr/apps/isv/matlab72/bin/matlab* to ease use for users. Because MATLAB incorporates a shell environment, `ssh` must be available for proper execution.

```
-l MATLAB=1, -v FEATURE=MATLAB=1, N1GE_MEM=2359296
```

Running the `n1ge matlab` command on the TSUBAME grid results in the following output:

```
tgg075003 admin/sun> n1ge matlab
*n1ge> Checking which tool have the command you specified....
*n1ge> Reading specific configuration file for each tools....
MATLAB needs arguments when you execute batch mode.
So MATLAB will run as foreground job.
*n1ge> Creating qrsh options....
*n1ge> Submitting Job to Cluster......
[Starting Job 344506 on tgg072188]
Warning: Unable to open display , MATLAB is starting without a display.
  You will not be able to display graphics on the screen.
Warning:
  MATLAB is starting without a display, using internal event queue.
  You will not be able to display graphics on the screen.

< M A T L A B >
Copyright 1984-2006 The MathWorks, Inc.
Version 7.2.0.283 (R2006a)
January 27, 2006

  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

>>
>> version

ans =
7.2.0.283 (R2006a)

>> exit
[Ending Job 344506]
Connection to tgg072188 closed.
tgg075003 admin/sun>
```

Applications and Voltaire MPI

Many jobs and user programs, such as Amber, POV-Ray, and GROMACS, use Voltaire MPI for parallelization. Table 7-1 lists the parallel environments for Voltaire MPI in the grid.

Table 7-1. Parallel environments for Voltaire MPI in the TSUBAME grid

vol_mpi_1p	vol_mpi_rr	vol_mpi_fillup	
vol_mpi_2p	vol_mpi_4p	vol_mpi_8p	vol_mpi_16p
vol_mpi_2t	vol_mpi_4t	vol_mpi_8t	vol_mpi_16t

Because the `n1ge` command assigns the parallel environment automatically, users do not need to be concerned with these parameters when submitting jobs. Example configurations for `vol_mpi_8p` and `vol_mpi_8t` are listed below.

Example `vol_mpi_8p` configuration:

```
pe_name            vol_mpi_8p
slots              20000
user_lists         NONE
xuser_lists        NONE
start_proc_args    /n1ge/TITECH_GRID/tools/pe/voltaire_mpi/startmpi.sh -catch_rsh  $pe_hostfile
stop_proc_args     /n1ge/TITECH_GRID/tools/pe/voltaire_mpi/stopmpi.sh
allocation_rule    8
control_slaves     TRUE
job_is_first_task FALSE
urgency_slots      min
```

Example `vol_mpi_8t` configuration:

```
pe_name            vol_mpi_8t
slots              20000
user_lists         NONE
xuser_lists        NONE
start_proc_args    /n1ge/TITECH_GRID/tools/pe/voltaire_mpi/startmpi_t.sh -catch_rsh  $pe_hostfile
stop_proc_args     /n1ge/TITECH_GRID/tools/pe/voltaire_mpi/stopmpi.sh
allocation_rule    8
control_slaves     TRUE
job_is_first_task FALSE
urgency_slots      min
```

The only difference between the `vol_mpi_8p` and `vol_mpi_8t` configurations is the setting of the shell script to `start_proc_args`, which creates the host list for `mpirun_ssh`. Because `vol_mpi_8t` is used for MPI and OpenMP programs, the `startmpi_t.sh` script creates the host list using the `$pe_hostfile` created by the Sun N1 Grid Engine software (Figure 7-1).

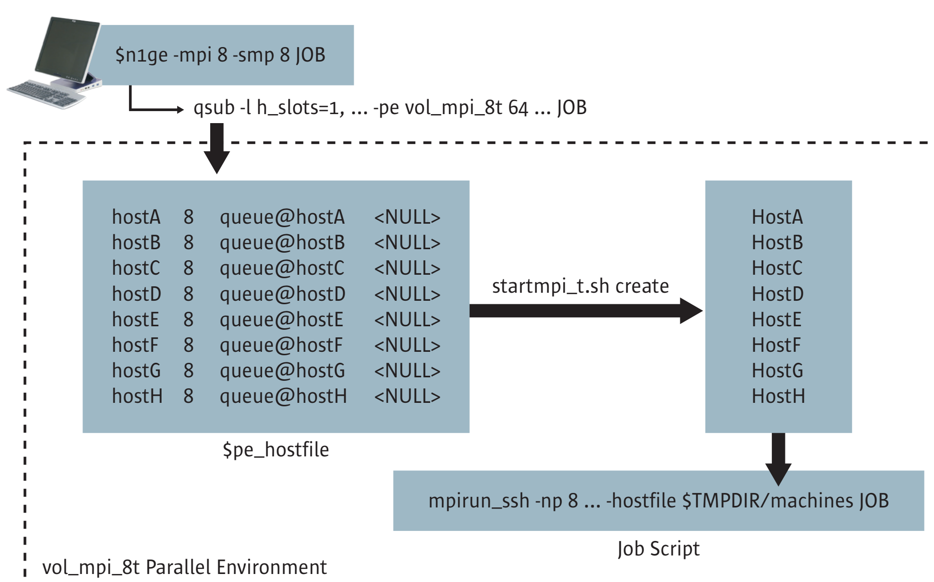

Figure 7-1. The host list creation process

User can specify OpenMP and MPI jobs by using the `-mpi` XX and `–smp` XX options to the `n1ge` command. If this case, the `n1ge` command creates the appropriate options for the `qsub` and `qrsh` commands using the process described in Figure 7-1. In this scenario, the job script uses the `$TMPDIR`*/machines* file. An example job script is listed below.

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
NCPUS=8
OMP_NUM_THREADS=8
export NCPUS OMP_NUM_THREADS
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/OTHERS.rc
MPIRUN=/usr/voltaire/mpi.pgcc.rsh/bin/mpirun_ssh
NP=8
PROG_ARGS=" a.out"
$MPIRUN -timeout 300 -ssh_cmd /n1ge/TITECH_GRID/tools/pe/voltaire/ssh \
-np $NP -hostfile $TMPDIR/machines $PROG_ARGS
echo [Ending Job $JOB_ID]
```

The startmpi.h Script

Running general Voltaire MPI applications requires writing a script which creates the host list as well as a wrapper script for the `qrsh` command. In addition, a job script must be created which uses `$TMPDIR/machines` as the host file and uses the `qrsh` wrapper script as an argument to the `–sshd_cmd` option. The key requirement of the `startmpil.sh` script is to transfer `$pe_hostfile` content to the host file. When users specify the `-pe` option, the Sun N1 Grid Engine software creates the following file when the `-pe vol_mpi_8p 120` option is specified. This file can be used as `$pe_hostfile` when configuring the parallel environment.

```
tgg072056 8 sla1@tgg072056 <NULL>
tgg072077 8 sla1@tgg072077 <NULL>
tgg072064 8 sla1@tgg072064 <NULL>
tgg072021 8 sla1@tgg072021 <NULL>
tgg072002 8 sla1@tgg072002 <NULL>
tgg072019 8 sla1@tgg072019 <NULL>
tgg072082 8 sla1@tgg072082 <NULL>
tgg072052 8 sla1@tgg072052 <NULL>
tgg072088 8 sla1@tgg072088 <NULL>
tgg072042 8 sla1@tgg072042 <NULL>
tgg072086 8 sla1@tgg072086 <NULL>
tgg072023 8 sla1@tgg072023 <NULL>
tgg072007 8 sla1@tgg072007 <NULL>
tgg072029 8 sla1@tgg072029 <NULL>
tgg072035 8 sla1@tgg072035 <NULL>
tgg072098 8 sla1@tgg072098 <NULL>
```

In this case, the value of `start_proc_args` is set to the following for the `vol_mpi_8p` and `vol_mpi_8t` parallel environments:

```
start_proc_args   /n1ge/TITECH_GRID/tools/pe/voltaire_mpi/startmpi.sh -catch_rsh $pe_hostfile
```

Wrapper Script for qrsh

It is important to integrate MPI programs with the Sun N1 Grid Engine software, and to be able to obtain CPU and memory usage for each process. The standard */usr/bin/ssh* command should not be used. Instead, use the `qrsh` wrapper script to run processes on the nodes in the host list. An example wrapper script used in the grid is listed below.

```
#!/bin/sh
#
could be rsh or remsh
me=`basename $0`
#just_wrap=1

#echo "Debug: command: $*"

# remove path to wrapping rsh from path list
PATH=`echo $PATH|tr : "Ä012"|grep -v $TMPDIR| tr "Ä012" :`
export PATH

# rehash
hash -r

if [ "x$JOB_ID" = "x" ]; then
   exec $me $*
   echo command $me not found in PATH=$PATH
fi

# extract target hostname
if [ $# -lt 1 ]; then
   echo $me: missing hostname
   exit 1
fi

# Handle hostname before options
rhost=
expr "$1" : "-*" >/dev/null 2>&1

if [ $? -ne 0 ]; then
   rhost=$1
   shift
fi

ruser=
minus_n=0
# parse other rsh options
while [ "$1" != "" ]; do
   case "$1" in
      -l)
   shift
      if [ $# -lt 1 ]; then
```

```
            echo $me: option -l needs user name as argument
            exit 1
         fi
         ruser=$1
         ;;
      -n)
         minus_n=1
         ;;
      -qq)
         cut_qq=1
         ;;
      *)
         break;
         ;;
   esac
   shift
done
# Handle hostname after options
if [ "x$rhost" = x ]; then
   if [ $# -lt 1 ]; then
      echo $me: missing hostname
      exit 1
   fi
   rhost=$1
   shift
fi
# should the command to be started preceeded with any starter command
#echo "Debug: RCMD_PREFIX = $RCMD_PREFIX"
if [ "x$RCMD_PREFIX" = x ]; then
   cmd="$*"
else
   cmd="$RCMD_PREFIX $*"
fi
# unset TASK_ID - it might be set if a task starts another tasks
#                 and may not be exported in this case
if [ "x$TASK_ID" = x ]; then
   unset TASK_ID
fi
if [ x$just_wrap = x ]; then
   #echo "Debug: rhost = $rhost"
   #echo "Debug: cmd = $cmd"
   if [ $minus_n -eq 1 ]; then
      #echo $SGE_ROOT/bin/$ARC/qrsh -V -inherit -nostdin $rhost $cmd
      exec $SGE_ROOT/bin/$ARC/qrsh -V -inherit -nostdin $rhost $cmd
   else
      #echo $SGE_ROOT/bin/$ARC/qrsh -V -inherit $rhost $cmd
      exec $SGE_ROOT/bin/$ARC/qrsh -V -inherit $rhost $cmd
   fi
else
   echo $me $rhost $*
   if [ $minus_n = 1 ]; then
      exec $me -n $rhost $cmd
   else
      exec $me $rhost $cmd
   fi
   echo $me not found in PATH=$PATH
fi
```

Job Script

The final task is to write the job script. An example job script is listed below. With Voltaire MPI, it is possible to specify the `ssh` command using the `mpirun_ssh` command with the `-ssh_cmd` option. Be sure to set the `PATH` as described below, and to use `$TMPDIR/machines` as an argument to the `-hostfile` option.

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/OTHERS.rc
MPIRUN=/usr/voltaire/mpi.pgcc.rsh/bin/mpirun_ssh
NP=$NSLOTS
PROG_ARGS="./a.out"
$MPIRUN -timeout 300 -ssh_cmd /n1ge/TITECH_GRID/tools/pe/voltaire/ssh \
-np $NP -hostfile $TMPDIR/machines $PROG_ARGS
echo [Ending Job $JOB_ID]
```

Fluent with Voltaire MPI

While Fluent can run with Voltaire MPI, there are some differences from the standard Voltaire MPI configuration.

1. Specify the *param.file* as an option to the `mpirun_ssh` command. Voltaire MPI prepares the parameter file for MPI, and it can be used as an argument to the `–paramfile` option of the `mpirun_ssh` command. Many Voltaire MPI parameters can be changed through the use of this file. Be sure to add the following to the bottom of the */usr/voltaire/mpi/doc/param.sample* file in order to run Fluent with Voltaire MPI.

```
LD_PRELOAD=/usr/lib64/libg2c.so.0
```

2. Set environment variables for Fluent, as listed below. Note that Fluent uses the `scp` command by default. However, the TSUBAME grid does not allow the use of the `scp` command between execution nodes. As a result, the TSUBAME grid configuration changes `RCOPY` to be set to `RCOPY=scp` in the Fluent script. While `TMP_RUN_SCRIPT` is not required, it is set to a common directory on all execution notes in the TSUBAME grid configuration. Finally, `IBA_MPIRUN` is set to the path of the `ssh` command.

```
IBA_MPIRUN="/usr/voltaire/mpi/bin/mpirun_ssh –paramfile $PARAM_FILE \
–ssh_cmd /n1ge/TITECH_GRID/tools/pe/voltaire/ssh"
IBA_VAPILIB="/usr/lib64:/usr/voltaire/lib"
TMP_RUN_SCRIPT="`pwd`/fluent-run-$$"
```

3. Specify the –`piba.voltaire` option on the `fluent` command line.
4. Specify `$TMPDIR/machines` as an argument of the –`cnf` option of the `fluent` command.

Example job script for Fluent:

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun
PATH=/home/admin/sun:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/FLUENT.rc
NP=$NSLOTS
/home3/usr6/ntsuzuki/bin/Fluent.Inc/bin/fluent 3ddp -piba.voltaire \
-ssh -t $NP -cnf=$TMPDIR/machines
echo [Ending Job $JOB_ID]
```

Example *FLUENT.rc* file contents:

```
FLUENT_INC="/home3/usr6/ntsuzuki/bin/Fluent.Inc"
PARAM_FILE="/n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_env/extend_files/
param.fluent"
IBA_MPIRUN="/usr/voltaire/mpi/bin/mpirun_ssh -paramfile $PARAM_FILE \
-ssh_cmd /n1ge/TITECH_GRID/tools/pe/voltaire/ssh"
IBA_VAPILIB="/usr/lib64:/usr/voltaire/lib"
TMP_RUN_SCRIPT="`pwd`/fluent-run-$$"

PATH=$FLUENT_INC/bin:$PATH

export FLUENT_INC IBA_MPIRUN IBA_VAPILIB TMP_RUN_SCRIPT PATH
```

Gaussian with TCP Linda

Running Gaussian with TCP Linda with the Sun N1 Grid Engine software several link0 commands must be placed in the input file, and environment variables must be defined.

- The `%NprocShared` Link0 command is the number of threads on each node. Set the value of `%NprocShared` to the number of CPUs allocated by the Sun N1 Grid Engine software. This command must be placed in the input file for Gaussian.
- The `%LindaWorker` Link0 command specifies the list of nodes, such as `hostA:1`. Set the value of `%LindaWorker` to the hostname assigned by the Sun N1 Grid Engine software. This command must be placed in the input file for Gaussian.
- Set the `GAUSS_LFLAGS` environment variable to the remote shell which with to fork *lXX.exe* on each node (`GAUSS_LFLAGS` = `-opt Tsnet.Node.lindarsharg:ssh`)
- Modify the *linda_rsh* file to use the wrapper script for the `qrsh` command.

Example parallel environment for TCP Linda:

```
pe_name           linda_8p
slots             20000
user_lists        NONE
xuser_lists       NONE
start_proc_args   /n1ge/TITECH_GRID/tools/pe/linda/startlinda.sh \
-catch_rsh $pe_hostfile
stop_proc_args    /n1ge/TITECH_GRID/tools/pe/linda/stoplinda.sh
allocation_rule   8
control_slaves    TRUE
job_is_first_task FALSE
urgency_slots     min
```

The *startlinda.sh* script creates the values of `%NprocShared` and `%LindaWorker` and places them in the *$TMPDIR/machines* file. An example *startlinda.sh* file is below.

```
#!/bin/sh
#
DATE=`date +"%Y/%m/%d %H:%M:%S"`
echo "Start Time => $DATE"
LOG_FILE=/n1ge/TITECH_GRID/tools/pe/linda/log/mpi.log

############################################################
# Logger
#
logging()
{
        arg=$@
        echo "`LANG=C date|awk '{print $2,$3,$4}'` $HOSTNAME: [ JOBID
$JOB_ID StartMPI ] $arg" >> $LOG_FILE
}

############################################################
# PeHostfile2LindaWorker()
#
PeHostfile2LindaWorker()
{
    lindaworker=""
    #cat $1 | while read line; do
    for line in `cat $1|cut -f1 -d" "|cut -f1 -d"."`
    do
        host=`echo $line|cut -f1 -d" "|cut -f1 -d"."`
        if [ "$lindaworker" = "" ]; then
            lindaworker="$host:1"
        else
            lindaworker="$lindaworker,$host:1"
        fi
    done
}
# parse options
catch_rsh=0
catch_hostname=0
unique=0
```

```
while [ "$1" != "" ]; do
   case "$1" in
      -catch_rsh)
         catch_rsh=1
         ;;
      -catch_hostname)
         catch_hostname=1
         ;;
      -unique)
         unique=1
         ;;
      *)
         break;
         ;;
   esac
   shift
done
me=`basename $0`

# test number of args
if [ $# -ne 1 ]; then
   echo "$me: got wrong number of arguments" >&2
   exit 1
fi

# get arguments
pe_hostfile=$1

# ensure pe_hostfile is readable
if [ ! -r $pe_hostfile ]; then
   echo "$me: can't read $pe_hostfile" >&2
   exit 1
fi
# Added by Hama
logging pe_hostfile is $pe_hostfile
echo "Master Host => $HOSTNAME"
echo "======================================="
echo "[The contents of pe_hostfile is below.]"
echo "---------------------------------------"
cat $pe_hostfile
echo ""

# create machine-file
# remove column with number of slots per queue
# mpi does not support them in this form
machines="$TMPDIR/machines"

# Set ARCH
ARCH=`$SGE_ROOT/util/arch`
# Find allocation rule
case "$PE" in
    linda_8p)    nprocshared=8    ;;
    linda_4p)    nprocshared=4    ;;
    linda_2p)    nprocshared=2    ;;
    linda_1p)    nprocshared=1    ;;
esac
```

```
PeHostfile2LindaWorker $pe_hostfile

# trace machines file
echo "================================================="
echo "[The NprocShared and LindaWorker will be following.]"
echo "-------------------------------------------------"
echo "%NprocShared=$nprocshared"
echo "%LindaWorker=$lindaworker"
echo ""
echo "%NprocShared=$nprocshared" > $machines
echo "%LindaWorker=$lindaworker" >> $machines
#
# Make script wrapper for 'rsh' available in jobs tmp dir
#
if [ $catch_rsh = 1 ]; then
   rsh_wrapper=/n1ge/TITECH_GRID/tools/pe/linda/ssh
   if [ ! -x $rsh_wrapper ]; then
      echo "$me: can't execute $rsh_wrapper" >&2
      echo "     maybe it resides at a file system not available at this
machine" >&2
      exit 1
   fi
   rshcmd=ssh
   ln -s $rsh_wrapper $TMPDIR/$rshcmd
fi

remain=`expr $JOB_ID % 10000`
target_dir=`expr $JOB_ID - $remain`
if [ -f "$SGE_STDOUT_PATH" ]; then
    cp $SGE_STDOUT_PATH  /n1ge/TITECH_GRID/job_info/$target_dir
fi

# signal success to caller
exit 0
```

If the `$pe_hostfile` file created by the Sun N1 Grid Engine software is as below, the *startlinda.sh* file creates a *$TMPDIR/machines* file with the following contents:

Example `$pe_hostfile`:

```
tgg075005 8 high@tgg075005 <NULL>
tgg075014 8 high@tgg075014 <NULL>
tgg075015 8 high@tgg075015 <NULL>
tgg075006 8 high@tgg075006 <NULL>
tgg075008 8 high@tgg075008 <NULL>
tgg075009 8 high@tgg075009 <NULL>
tgg075007 8 high@tgg075007 <NULL>
tgg075012 8 high@tgg075012 <NULL>
```

Example *$TMPDIR/machines* file:

```
%NprocShared=8
%LindaWorker=tgg075005:1,tgg075014:1,tgg075015:1,tgg075006:1,tgg075008:
1,tgg075009:1,tgg075007:1,tgg075012:1
```

The next task is to modify the *linda_rsh* file to use the `qrsh` wrapper script. The original *linda_rsh* file specifies */usr/bin/ssh* as the full path to the `ssh` command.

Original *linda_rsh* file:

```
.....
case "$rsh_arg" in
ssh) xonrsh=/usr/bin/ssh
      ;;
*)  xonrsh=/usr/bin/rsh
     ;;
esac
.....
   *)   case "$rsh_arg" in
           on) exec /usr/bin/on -n $host "$@"
               ;;
           ssh) exec /usr/bin/ssh -x $host $user -n "$@"
               ;;
           *)  exec /usr/bin/rsh $host $user -n "$@"
               ;;
         esac
         ;;
.....
```

The *startlinda.sh* script sets a symbolic link for the `qrsh` wrapper script to `$TMPDIR/ssh` and sets the `PATH` to `$TMPDIR`. As a result, TCP Linda can use the `qrsh` wrapper script as the `ssh` command in the event a full path is not specified for the remote shell. The *linda_rsh* file should be modified per the listing below. Note that `numactl` is optional, if it is acceptable to treated memory as interleaved in Gaussian. In addition, be sure to change the arguments to the `-c` and `-i` options to reflect the hardware environment.

```
.....
case "$rsh_arg" in
ssh) xonrsh=ssh
      ;;
*)  xonrsh=/usr/bin/rsh
     ;;
esac
.....
   *)   case "$rsh_arg" in
           on) exec /usr/bin/on -n $host "$@"
               ;;
           ssh) exec ssh $host $user -n "numactl -c 6,2,3,7,5,1,0,4 -i 6,2,3,7,5,1,0,4 $@"
               ;;
           *)  exec /usr/bin/rsh $host $user -n "$@"
               ;;
         esac
         ;;
.....
```

The final task is to write a job script, as well as a script which modifies the input file for Gaussian to use the `%NprocShared` and `%LindaWorker` commands. The following job script file, created automatically by the `n1ge` command, enables Gaussian to run with TCP Linda on the Sun N1 Grid Engine software.

```
#!/bin/sh
#$ -S /bin/sh
cd /n1ge/TITECH_GRID/tools/pe/linda
PATH=/n1ge/TITECH_GRID/tools/pe/linda:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/Gaussian.Linda.rc
cat Default.Route
if [ $RESTARTED = 1 ]; then
    echo "Checking if input file exists because this job was re-queued."
    if [ -r "test397_LindaWorker.com.$JOB_ID" ]; then
        mv test397_LindaWorker.com.$JOB_ID test397_LindaWorker.com
    else
        echo "The original input file test397_LindaWorker.com.$JOB_ID
doesn't exist."
        echo "Stopping this job for safety."
        exit 1
    fi
fi
echo "Saving original input file as test397_LindaWorker.com.$JOB_ID"
/n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_env/extend_files/
mod_gau_input.pl \
test397_LindaWorker.com
time numactl -c 6,2,3,7,5,1,0,4 -i 6,2,3,7,5,1,0,4 /usr/apps/isv/
gaussian_linda/g03/g03 Ä
test397_LindaWorker.com
if [ -r "test397_LindaWorker.com.$JOB_ID" ]; then
    echo "Restoring original input file from
test397_LindaWorker.com.$JOB_ID"
    mv test397_LindaWorker.com.$JOB_ID test397_LindaWorker.com
fi
echo [Ending Job $JOB_ID]
```

The *Gaussian.Linda.rc* file sets the environment variables for Gaussian. The *mod_gau_input.pl* script creates a new input file for Gaussian that uses `$TMPDIR/machines` with the value of the `%NprocShared` and `%LindaWorker` commands. The following example *Gaussian.Linda.rc* and *mod_gau_input.pl* files are used in the TSUBAME supercomputer grid.

```
g03root="/usr/apps/isv/gaussian_linda"
PGI="/usr/apps/isv/pgi"
GAUSS_SCRDIR=/gau/$HOSTNAME
GAUSS_LFLAGS='-opt "Tsnet.Node.lindarsharg:ssh"'
if [ ! -d "$GAUSS_SCRDIR" ]; then
    echo "Can't find $GAUSS_SCRDIR"
    echo "Exiting...with 99"
    exit 99
fi
#
# g03.profile
#
gr=$g03root
GAUSS_EXEDIR="$gr/g03/bsd:$gr/g03/private:$gr/g03"
GAUSS_LEXEDIR="$gr/g03/linda-exe"
GAUSS_ARCHDIR="$gr/g03/arch"
GMAIN=$GAUSS_EXEDIR
PATH=$GAUSS_EXEDIR:$PATH
LD_LIBRARY_PATH=$GAUSS_EXEDIR:$LD_LIBRARY_PATH
G03BASIS="$gr/g03/basis"
F_ERROPT1="271,271,2,1,2,2,2,2"
#following for sgi debugging
#TRAP_FPE="DEBUG;OVERFL=ABORT;DIVZERO=ABORT;INVALID=ABORT;INT_OVERFL=ABORT"
TRAP_FPE="OVERFL=ABORT;DIVZERO=ABORT;INT_OVERFL=ABORT"
MP_STACK_OVERFLOW="OFF"
# to partially avoid KAI stupidity
KMP_DUPLICATE_LIB_OK="TRUE"
export GAUSS_EXEDIR GAUSS_ARCHDIR PATH GMAIN LD_LIBRARY_PATH F_ERROPT1
TRAP_FPE MP_STACK_OVERFLOW \
  KMP_DUPLICATE_LIB_OK G03BASIS GAUSS_LEXEDIR GAUSS_LFLAGS
```

The TSUBAME supercomputer grid uses the Lustre file system for the Gaussian scratch directory. If a job cannot find the Gaussian scratch directory, */gau/hostname*, the job exits with a status code of 99 and is re-queued automatically by the Sun N1 Grid Engine software.

Example *mod_gau_input.pl* file:

```
#!/usr/bin/perl
#
# mod_gau_input.pl
#
#       version 0.1
#               First creation
#               by Sun Microsystems.K.K. 05/05/2006
#
use File::Copy;

##############################################################################
# Setting Variables
#
if (@ARGV == 0) {
    print STDERR "ERROR: Please specify Gaussian input file.Än";
    print STDERR "ERROR: Please contact system administratorÄn";
}
my $GAU_INPUT = $ARGV[0];
if ( $GAU_INPUT eq "<" ) {
    $GAU_INPUT = $ARGV[1];
}
chomp(@line = `pwd`);
my $CWD = $line[0];

###############################################################################
#
# Main
#
unless ( -r "$CWD/$GAU_INPUT" ) {
    print STDERR "ERROR: Can't open $CWD/$GAU_INPUTÄn";
    print STDERR "ERROR: Please contact system administratorÄn";
}

#
my $JOB_ID = "$ENV{'JOB_ID'}";
my $TMPDIR = "$ENV{'TMPDIR'}";
my $GAU_SAVE = "$CWD/$GAU_INPUT.$JOB_ID";
my $ADD_CONTENTS = "$TMPDIR/machines";

#
open(ADD_CONTENTS, "$ADD_CONTENTS");
my @add_contents = <ADD_CONTENTS>;
close ADD_CONTENTS;

#
move("$GAU_INPUT","$GAU_SAVE") or die;
open(INPUT_FILE, "$GAU_SAVE");
my @original = <INPUT_FILE>;
close INPUT_FILE;
open(NEW_FILE, ">> $GAU_INPUT");
print NEW_FILE @add_contents;           # contents of machines file
print NEW_FILE @original;               # contents of original input file
close NEW_FILE;
```

GAMESS and the Distributed Data Interface

GAMESS uses the Distributed Data Interface for parallelization. In order to run GAMESS and the Distributed Data Interface with the Sun N1 Grid Engine software, several environment variables must be set, including:

- NNODES, the number of nodes on which to run GAMESS
- HOSTLIST, a list of node with the format hostA:cpus=16
- DDI_RSH, the remote shell command which forks GAMESS not on the master host
- SCR, the scratch directory

To run Gaussian and DDI with the Sun N1 Grid Engine software:

1. Set the value of NNODE and HOSTLIST using the number of CPUs and the hostname allocated by the Sun N1 Grid Engine software.
2. Modify the rungms command.

Example parallel environment for DDI:

```
pe_name           ddi_16p
slots             20000
user_lists        NONE
xuser_lists       NONE
start_proc_args   /n1ge/TITECH_GRID/tools/pe/ddi/startddi.sh \
-catch_rsh $pe_hostfile
stop_proc_args    /n1ge/TITECH_GRID/tools/pe/ddi/stopddi.sh
allocation_rule   16
control_slaves    TRUE
job_is_first_task FALSE
urgency_slots     min
```

The *startddi.sh* script creates the values for the NODES and HOSTLIST environment variables, and places the values in the *$TMPDIR/machines* file. The *startddi.sh* file used in the TSUBAME grid is displayed below.

```
#!/bin/sh
#
DATE=`date +"%Y/%m/%d %H:%M:%S"`
echo "Start Time => $DATE"
LOG_FILE=/n1ge/TITECH_GRID/tools/pe/ddi/log/mpi.log

############################################################
# Logger
#
logging()
{
        arg=$@
        echo "`LANG=C date|awk '{print $2,$3,$4}'` $HOSTNAME: [ JOBID
$JOB_ID StartMPI ] $arg" >> $LOG_FILE
}
```

```
############################################################
# PeHostfile2LindaWorker()
#
PeHostfile2HOSTLIST()
{
    HOSTLIST=""
    num_host=0
    while read line
    do
        num_host=`expr $num_host + 1`
        host=`echo $line|cut -f1 -d" "|cut -f1 -d"."`
        num_cpu=`echo $line|cut -f2 -d" "`
        if [ "$HOSTLIST" = "" ]; then
            HOSTLIST="$host:cpus=$num_cpu"
        else
            HOSTLIST="$HOSTLIST $host:cpus=$num_cpu"
        fi
    done < $1
}

# parse options
catch_rsh=0
catch_hostname=0
unique=0
while [ "$1" != "" ]; do
   case "$1" in
      -catch_rsh)
         catch_rsh=1
         ;;
      -catch_hostname)
         catch_hostname=1
         ;;
      -unique)
         unique=1
         ;;
      *)
         break;
         ;;
   esac
   shift
done
me=`basename $0`

# test number of args
if [ $# -ne 1 ]; then
   echo "$me: got wrong number of arguments" >&2
   exit 1
fi

# get arguments
pe_hostfile=$1

# ensure pe_hostfile is readable
if [ ! -r $pe_hostfile ]; then
   echo "$me: can't read $pe_hostfile" >&2
   exit 1
fi
```

```
# Added by Hama
logging pe_hostfile is $pe_hostfile
echo "Master Host => $HOSTNAME"
echo "========================================"
echo "[The contents of pe_hostfile is below.]"
echo "----------------------------------------"
cat $pe_hostfile
echo ""

# create machine-file
# remove column with number of slots per queue
# mpi does not support them in this form
machines="$TMPDIR/machines"

# Set ARCH
ARCH=`$SGE_ROOT/util/arch`

PeHostfile2HOSTFILE $pe_hostfile

# trace machines file
echo "================================================="
echo "[The NprocShared and LindaWorker will be following.]"
echo "-------------------------------------------------"
echo "NNODES=$num_host"
echo "HOSTLIST=$HOSTLIST"
echo ""
echo "NNODES=$num_host" > $machines
echo "HOSTLIST=$HOSTLIST" >> $machines
#
# Make script wrapper for 'rsh' available in jobs tmp dir
#
if [ $catch_rsh = 1 ]; then
   rsh_wrapper=/n1ge/TITECH_GRID/tools/pe/ddi/ssh
   if [ ! -x $rsh_wrapper ]; then
      echo "$me: can't execute $rsh_wrapper" >&2
      echo "     maybe it resides at a file system not available at this
machine" >&2
      exit 1
   fi
   rshcmd=ssh
   ln -s $rsh_wrapper $TMPDIR/$rshcmd
fi

remain=`expr $JOB_ID % 10000`
target_dir=`expr $JOB_ID - $remain`
if [ -f "$SGE_STDOUT_PATH" ]; then
    cp $SGE_STDOUT_PATH  /n1ge/TITECH_GRID/job_info/$target_dir
fi

# signal success to caller
exit 0
```

If the `$pe_hostfile` file created by the Sun N1 Grid Engine software is as below, the *startddi.sh* file creates a *$TMPDIR/machines* file with the following contents:

Example *$pe_hostfile*:

```
tgg073211 8 B@tgg073211 <NULL>
tgg073184 8 B@tgg073184 <NULL>
tgg073231 8 B@tgg073231 <NULL>
tgg073128 8 B@tgg073128 <NULL>
```

Example *$TMPDIR/machines* file:

```
NNODES=4
HOSTLIST=tgg073211:cpus=8 tgg073184:cpus=8 tgg073231:cpus=8
tgg073128:cpus=8
```

Modify the rungms Command

The next task is to modify the `rungms` command for the environment. An example from the TSUBAME grid is displayed below.

```
.....
set TARGET=sockets
set check_scr=${?SCR}
if ($check_scr == 0) then
    echo "Please set "\$SCR" directory."
    exit 1
endif
#
set JOB=$1       # name of the input file xxx.inp, give only the xxx part
set VERNO=00     # revision number of the executable created by 'lked' step
set NCPUS=$2     # number of compute processes to be run
.....
```

Create the Job Script

The last task is to create the job script for the Sun N1 Grid Engine software that enables GAMESS to run with DDI. This script is created automatically by the n1ge command.

Example job script:

```
#!/bin/sh
#$ -S /bin/sh
cd /home/admin/sun/hama/gamess
PATH=/home/admin/sun/hama/gamess:$PATH
export PATH
echo [Starting Job $JOB_ID on $HOSTNAME]
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/apps.sh
. /n1ge/TITECH_GRID/tools/bin/n1ge_bin/apps_rc/GAMESS.rc
/usr/apps/free/gamess/rungms ericfmt.dat $NSLOTS
echo [Ending Job $JOB_ID]
```

Creating the .rc File and Setting Environment Variables

The *GAMESS.rc* file used in the TSUBAME grid can be used as an example for creating *GAMESS.rc* files for other environments.

```
GMSPATH=/usr/apps/free/gamess
if [ "$SCR" = "" ]; then
    SCR=`pwd`
fi
PATH=$GMSPATH:$PATH
DDI_RSH=/n1ge/TITECH_GRID/tools/pe/ddi/ssh
export GMSPATH PATH DDI_RSH SCR
if [ -r $TMP/machines ]; then
    while read line
    do
        export "$line"
    done < $TMP/machines
    echo "NNODES => $NNODES"
    echo "HOSTLIST => $HOSTLIST"
fi
```

User should specify the `SCR` environment variable prior to submitting GAMESS jobs with the `n1ge` command. By setting the `SCR` environment variable, GAMESS uses the */work/games* directory as the scratch directory.

```
$ export SCR=/work/gamess
$ n1ge –ddi 64 –mem 2 rungms input.file
```

Chapter 8
For More Information

About the Author

Minoru Hamakawa is a consultant in Sun's Professional Services organization. Since joining Sun in 2001, Minoru has worked in a variety of consulting roles aimed at helping customers create the most effective computing environments. Since 2002, Minoru has consulted on a variety of customer challenges in high performance computing environments. His work on the Tokyo Tech TSUBAME supercomputer grid focused on application and operating system configuration on the execution nodes in the grid, as well as the deployment and configuration of the Sun N1 Grid Engine software.

Acknowledgements

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

- Professor Satoshi Matsuoka of the Tokyo Institute of Technology, for his advice on TSUBAME grid architecture and performance.
- Yifta Shahar of Voltaire Inc., for his advice on Infiniband networking.
- Friedrich Ferstl, Andreas Schwierskott and the rest of the Sun N1 Grid Engine software engineering team, for their support on the Sun N1 Grid Engine system architecture.
- Marshall Choy, for providing insight into the Sun x64 server architecture.
- Junichi Koike, for leading the TSUBAME grid architecture design effort and providing advice on the implementation.
- Toshihiro Kujiraoka, for leading the TSUBAME grid project and providing tremendous support throughout system delivery.
- All other individuals involved in the TSUBAME grid project.

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 (9786) **Web** sun.com

© 2007 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, Sun BluePrints, N1, and Sun Fire are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Information subject to change without notice.

Printed in USA 06/07