UNIVA

Univa Corporation

Univa Grid Engine Documentation

Univa Grid Engine Administrator's Guide

Author:
Univa Engineering

Version:
8.6.3

September 27, 2018

Copyright ©2012–2018 Univa Corporation. All rights reserved.

Contents

1 Navigating and Understanding **1**

1.1 Navigating the Univa Grid Engine System 1

1.1.1 Location of Univa Grid Engine Configuration Files and Binaries 1

1.1.2 Displaying Status Information 8

1.2 Understanding a Default Installation 14

1.2.1 Default Queue 14

1.2.2 Default PE 16

1.2.3 Default User Set Lists 16

1.2.4 Default Host Group List 16

1.2.5 Default Complex Attributes 17

1.3 Understanding Key Univa Grid Engine Configuration Objects 18

1.3.1 The Cluster Configuration 18

1.3.2 The Scheduler Configuration 18

1.3.3 Host and Queue Configurations 19

1.4 Navigating the ARCo Database 19

1.4.1 Accessing the ARCo Database 19

1.4.2 Views to the Database 19

1.4.3 Database Tables 33

2 Common Tasks **49**

2.1 Common Administrative Tasks in a Univa Grid Engine System 49

2.1.1 Draining Then Stopping the Cluster 49

2.1.2 Starting Up and Activating Nodes Selectively 50

2.1.3 Adding New Execution Hosts to an Existing Univa Grid Engine System . 51

2.1.4 Generate/Renew Certificates and Private Keys for Users 52

2.1.5 Backup and Restore the Configuration 54

2.1.6 Changing the Univa Grid Engine admin password for all UGE Starter Services on all execution hosts 57

2.2 Managing User Access 58

2.2.1 Setting Up a Univa Grid Engine User 59

2.2.2 Managers 60

2.2.3 Operators and Owners 60

2.2.4 Permissions of Managers, Operators, Job or Queue Owners 61
2.2.5 User Access Lists and Departments 63
2.2.6 Projects 65
2.3 Understanding and Modifying the Cluster Configuration 66
2.3.1 Commands to Add, Modify, Delete or List Global and Local Configurations 66
2.3.2 Configuration Parameters of the Global and Local Configurations 67
2.4 Understanding and Modifying the Univa Grid Engine Scheduler Configuration . 69
2.4.1 The Default Scheduling Scheme 69
2.5 Configuring Properties of Hosts and Queues 71
2.5.1 Configuring Hosts 72
2.5.2 Configuring Queues 77
2.5.3 Utilizing *Complexes* and *Load Sensors* 83
2.5.4 Configuring and Using the RSMAP Complex Type 89
2.5.5 Advanced Attribute Configuration 95
2.5.6 Configuring and Using Linux cgroups 97
2.6 Monitoring and Modifying User Jobs 102
2.7 Diagnostics and Debugging 102
2.7.1 KEEP_ACTIVE functionality 102
2.7.2 Diagnosing Scheduling Behavior 103
2.7.3 Location of Logfiles and Interpreting Them 104
2.7.4 Turning on Debugging Information 106

3 Special Activities **114**
3.1 Tuning Univa Grid Engine for High Throughput 114
3.1.1 sge_qmaster Tuning 114
3.1.2 Tuning Scheduler Performance 116
3.1.3 Reducing Overhead on the Execution Side 116
3.2 Optimizing Utilization 117
3.2.1 Using Load Reporting to Determine Bottlenecks and Free Capacity 117
3.2.2 Scaling the Reported Load 119
3.2.3 Alternative Means to Determine the Scheduling Order 121
3.3 Managing Capacities 124
3.3.1 Using *Resource Quota Sets* 124
3.3.2 Using *Consumables* 126

3.4 Implementing Pre-emption Logic . . . 131
3.4.1 When to Use Pre-emption . . . 131
3.4.2 Utilizing Queue Subordination . . . 131
3.4.3 Advanced Pre-emption Scenarios . . . 132
3.5 Integrating Univa Grid Engine With a License Management System . . . 134
3.5.1 Integrating and Utilizing QLICSERVER . . . 135
3.6 Managing Priorities and Usage Entitlements . . . 136
3.6.1 Share Tree (Fair-Share) Ticket Policy . . . 136
3.6.2 Functional Ticket Policy . . . 148
3.6.3 Override Ticket Policy . . . 149
3.6.4 Handling of Array Jobs with the Ticket Policies . . . 150
3.6.5 Urgency Policy . . . 151
3.6.6 User Policy: POSIX Policy . . . 156
3.7 Job Placement . . . 157
3.7.1 Host/Queue Sorting . . . 157
3.7.2 Affinity, Anti-Affinity, Best Fit . . . 160
3.8 Advanced Management for Different Types of Workloads . . . 163
3.8.1 Parallel Environments . . . 163
3.8.2 Setting Up Support for Interactive Workloads . . . 169
3.8.3 Setting Up Support for Checkpointing Workloads . . . 170
3.8.4 Enabling Reservations . . . 172
3.8.5 Greedy Resource Reservation . . . 176
3.8.6 Simplifying Job Submission Through the Use of Default Requests . . . 181
3.8.7 Job Submission Verifiers . . . 182
3.8.8 Enabling and Disabling Core Binding . . . 198
3.9 Ensuring High Availability . . . 198
3.9.1 Prerequisites . . . 199
3.9.2 Installation . . . 199
3.9.3 Testing sge_shadowd Takeover . . . 199
3.9.4 Migrating the Master Host Back After a Takeover . . . 200
3.9.5 Tuning the sge_shadowd . . . 200
3.9.6 Troubleshooting . . . 201
3.10 Utilizing Calendar Schedules . . . 201

3.10.1 Commands to Configure Calendars . . . 202
3.10.2 Calendars Configuration Attributes . . . 202
3.10.3 Examples to Illustrate the use of Calendars . . . 204
3.11 Setting Up Nodes for Exclusive Use . . . 205
3.12 Deviating from a Standard Installation . . . 206
3.12.1 Utilizing Cells . . . 206
3.12.2 Using Path Aliasing . . . 206
3.12.3 Host-name Resolving and Host Aliasing . . . 207
3.13 Using CUDA Load Sensor . . . 209
3.13.1 Building the CUDA load sensor . . . 210
3.13.2 Installing the load sensor . . . 211
3.13.3 Using the CUDA load sensor . . . 212
3.14 Support of Intel® Xeon Phi™ Co-Processors . . . 213
3.14.1 The Intel Xeon Phi Load Sensor . . . 213
3.14.2 Exploiting the RSMAP topology mask with Intel® Xeon Phi™ Co-Processors 218
3.14.3 Submission and Directly Starting of an Intel Xeon Phi (MIC) Native Application . . . 221
3.14.4 The mic_load_check tool . . . 222
3.14.5 Configure and Install Intel Xeon Phi x200 (Knights Landing) Processors support . . . 223
3.14.6 Installing Intel Xeon Phi x200 (Knights Landing) Processors support . . . 223
3.15 Integration with Docker® Engine . . . 224
3.15.1 Docker images suitable for autostart Docker jobs with arguments . . . 225
3.15.2 Run the container as root, allow to run prolog etc. as a different user . . . 226
3.15.3 Automatically map user ID and group ID of a user into the container . . . 226
3.15.4 Create a container_pe_hostfile with all container hostnames . . . 227
3.15.5 Run tightly integrated parallel jobs in Docker containers . . . 227
3.15.6 Configuring the Docker daemon response timeout . . . 230
3.16 Special Tools . . . 230
3.16.1 The Loadcheck Utility . . . 230
3.16.2 Utilities for BDB spooling . . . 231

1 Navigating and Understanding

1.1 Navigating the Univa Grid Engine System

Univa Grid Engine consists of different modules, which are usually distributed over a large number of hosts. This chapter provides a high level overview of a Univa Grid Engine installation, including details on where the execution binaries and configuration files are located, how status information about different components and objects can be displayed, and how they are interpreted.

1.1.1 Location of Univa Grid Engine Configuration Files and Binaries

To interact with a Univa Grid Engine, the client binaries and basic configuration parameters must be available in the shell environment. To do the whole shell environment setup, simply source a pre-defined shell script generated during the product installation. One major part of the working environment is the `$SGE_ROOT` environment variable, which contains the full path to the Univa Grid Engine installation. Using such environment variables allows interactions with different Univa Grid Engine installations on the same host.

The following example assumes that Univa Grid Engine is installed in the `/opt/UGE820` directory, and that the user works with `bash`. This example shows how the environment of this particular installation is sourced in order to interact with the system.

```
> source /opt/UGE820/default/common/setting.sh
```

Within a C-shell the corresponding settings script must be sourced:

```
> source /opt/UGE820/default/common/setting.csh
```

And on Windows (win-x86), it is assumed the directory of Univa Grid Engine is available on \\fileserver\share\opt\UGE820. There, the following batch script must be executed:

```
>  \\fileserver\share\opt\UGE820\default\common\settings.bat
```

Environment Variable	Description
`$SGE_ROOT`	The absolute path to the Univa Grid Engine product installation.
`$ARCH`	The Univa Grid Engine architecture string. It identifies the OS and in some cases the processor architecture. This variable is not set on Windows (win-x86).
`$SGE_CELL`	The name of the Univa Grid Engine cell. The purpose of the cell name is to distinguish different clusters, which are using the same binary installation (and therefore having the same `$SGE_ROOT`).
`$SGE_CLUSTER_NAME`	The system wide unique name of the Univa Grid Engine cluster.
`$SGE_QMASTER_PORT`	Network port where the master daemon is listening.

Environment Variable	Description
`$SGE_EXECD_PORT`	Network port where the execution daemons are listening.
`$PATH`	The default path variable is extended with the path to the Univa Grid Engine binary directory.
`$MANPATH`	The manual pages path variable is extended in order to provide command line access to the various Univa Grid Engine man pages. This variable is not set on Windows (win-x86).
library path	Path to Univa Grid Engine libraries. Only set on architectures that do not have a build-in run-path. The library path variable depends on the OS type.

Table 1: Environment Variables Set by the `setting` Script

The following graphic illustrates the structure of the `$SGE_ROOT` directory.

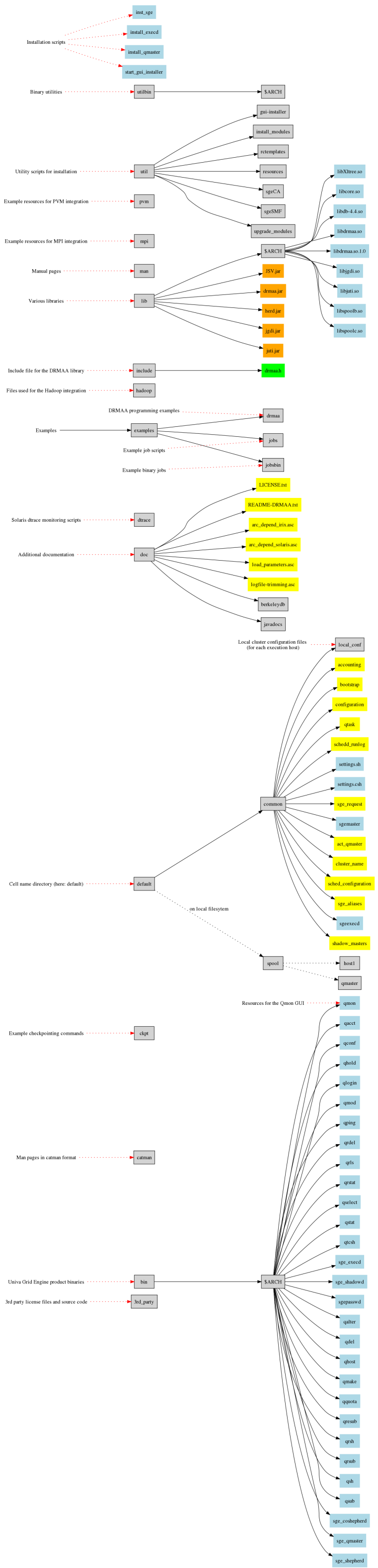

inst_sge
install_execd
install_qmaster
start_gui_installer
Installation scripts
Binary utilities
utilbin
$ARCH
gui-installer
install_modules
rctemplates
resources
sgeCA
sgeSMF
upgrade_modules
Utility scripts for installation
util
Example resources for PVM integration
pvm
Example resources for MPI integration
mpi
libXltree.so
libcore.so
libdb-4.4.so
libdrmaa.so
libdrmaa.so.1.0
libjgdi.so
libjuti.so
libspoolb.so
libspoolc.so
$ARCH
JSV.jar
drmaa.jar
herd.jar
jgdi.jar
juti.jar
Manual pages
man
Various libraries
lib
Include file for the DRMAA library
include
drmaa.h
Files used for the Hadoop integration
hadoop
DRMAA programming examples
drmaa
Examples
examples
jobs
Example job scripts
jobsbin
Example binary jobs
LICENSE.txt
README-DRMAA.txt
arc_depend_irix.asc
arc_depend_solaris.asc
load_parameters.asc
logfile-trimming.asc
berkeleydb
javadocs
Solaris dtrace monitoring scripts
dtrace
Additional documentation
doc
Local cluster configuration files
(for each execution host)
local_conf
accounting
bootstrap
configuration
qtask
schedd_runlog
settings.sh
settings.csh
sge_request
sgemaster
act_qmaster
cluster_name
sched_configuration
sge_aliases
sgeexecd
shadow_masters
common
Cell name directory (here: default)
default
on local filesytem
spool
host1
qmaster
Resources for the Qmon GUI
qmon
Example checkpointing commands
ckpt
qacct
qconf
qhold
qlogin
qmod
qping
qrdel
qrls
qrstat
qselect
qstat
qtcsh
sge_execd
Man pages in catman format
catman
Univa Grid Engine product binaries
bin
$ARCH
sge_shadowd
3rd party license files and source code
3rd_party
sgepasswd
qalter
qdel
qhost
qmake
qquota
qresub
qrsh
qrsub
qsh
qsub
sge_coshepherd
sge_qmaster
sge_shepherd

FIGURE 1: Overview of the $SGE_ROOT Directory Structure

The main configuration files are located in the `$SGE_ROOT/$SGE_CELL/common` directory (represented by the yellow files in Figure 1 above). Most of these files are generated automatically when the configuration is changed by the corresponding Univa Grid Engine administration command. The following table provides an overview of these files.

Configuration File	Description
`accounting`	Contains accounting information about past jobs. The `qacct` client reads this data.
`bootstrap`	Contains information about spooling and multi-threading for the `qmaster`.
`configuration`	Contains the current global cluster configuration, which can be modified by the `qconf -mconf` command.
`qtask`	The qtask configuration file (see man page `qtask`).
`schedd_runlog`	Contains information about a scheduling run, when it is monitored (see `qconf -tsm`).
`cluster_name`	Contains the unique cluster name.
`sched_configuration`	Contains the current scheduler configuration, which can be modified by the `qconf -tsm` command.
`sge_aliases`	The path aliasing configuration file.
`shadow_masters`	Contains a list of shadow daemons.
`local_conf/<hostname>`	All files in this directory represent the local cluster configuration for the specific host, and they can be modified by the `qconf -mconf <hostname>` command.
`path_map`	Exists only if Windows hosts are in the cluster. Contains the mapping between Unix paths and the corresponding Windows (win-x86) paths.

Table 2: Overview of Main Configuration Files

During run-time, all scheduler decisions and status information are written to files (classic spooling) or a database (BDB spooling), either of which is usually held on a secondary storage (like fast SSDs, and/or hard drives). This is done so that, in case of problems, newly started daemons can retrieve the current state and can immediately proceed with operation. There are two types of spooling directories, one for the `master daemon` and one for the `execution daemon`. The execution daemon spooling directories should point to a local directory (and not a NFS shared directory) for the performance benefit. For Windows execution daemons, the spooling directory must point to a local directory. When `shadow daemon` is configured, the master spooling directory must be shared with the shadow daemon host; if not, the master spooling should also be held locally.

Qmaster Spooling Directory

The following graphics illustrate the structure of the qmaster spooling directory (Figure 2):

FIGURE 2: Overview of the qmaster Spooling Directory

Execution Host Spooling Directory

The following graphics illustrate the structure of the execution host spooling directory (Figure 3 to 6):

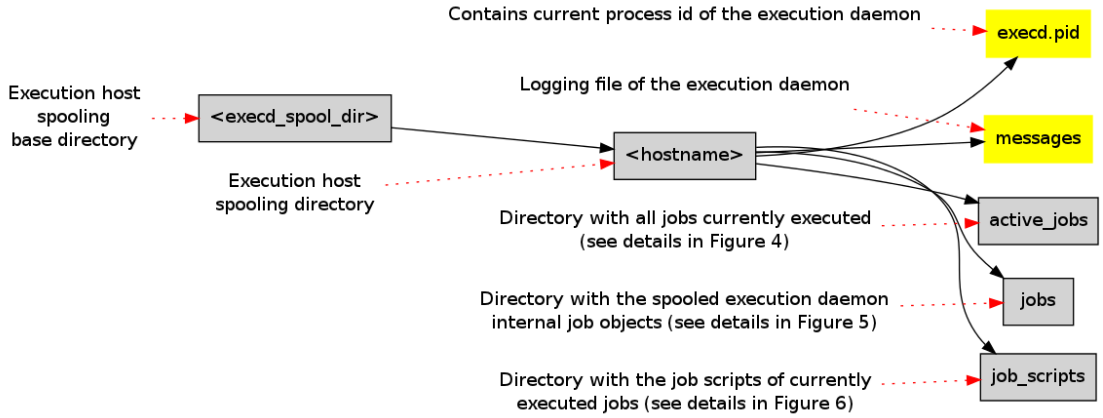


FIGURE 3: Overview of the Execution Host Spooling Directory

Directory or File	Description
`<execd_spool_dir>`	The execution host spooling base directory as defined by the configuration value "execd_spool_dir" in the execution host or global host configuration.
`<hostname>`	The execution host specific spooling subdirectory. The name of this directory is the name of the execution host.
`execd.pid`	The process ID of the execution daemon. It writes this file after start.
`messages`	The log file of the execution daemon. The amount of messages logged to this file depends on the configuration value "loglevel".
`active_jobs`	In this subdirectory, the execution daemon creates a directory for each task of the job that is to be started on this execution host.
`jobs`	In this subdirectory, the execution daemon spools the job objects it uses internally.
`job_scripts`	In this subdirectory, the execution daemon stores the job scripts of all jobs that have tasks that are to be started on this execution host.

Table 3: Execution Host Spooling Directory Details

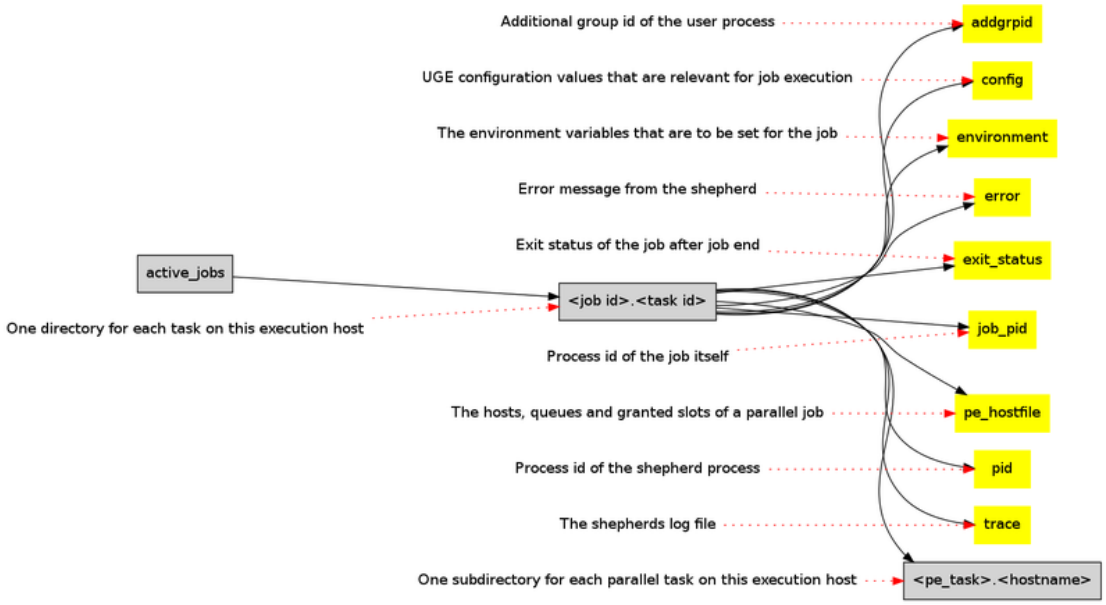


FIGURE 4: Overview of the Active Jobs Spooling Sub Directory

Directory or File	Description
`<job id>.<task id>`	For each task of a job such a subdirectory is created.
`<pe_task_id>.<hostname>`	For each parallel task of a tightly integrated parallel job, the execution daemon creates such a subdirectory right before it starts the parallel task. This subdirectory contains the same files as the “active_jobs” directory, except for the “pe_hostfile”.

Table 4: Active Jobs Subdirectory Details

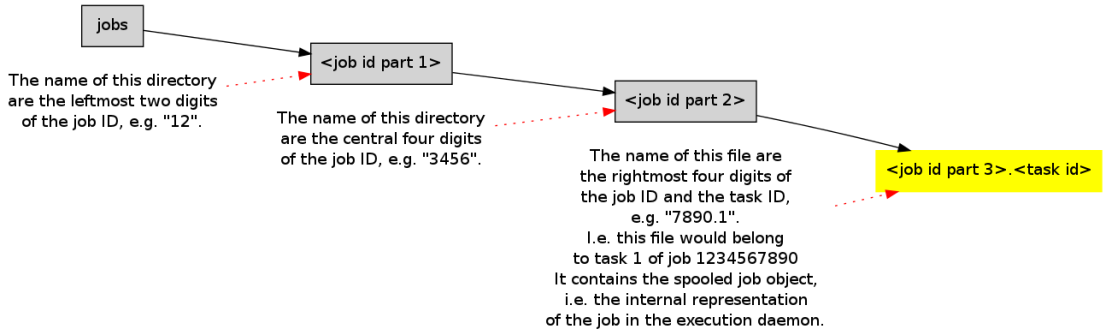


FIGURE 5: Overview of the Jobs Spooling Sub Directory

The jobs spooling directory is split up into this structure because most filesystems become slow when there are too many files or subdirectories in one directory. This wouldn’t be a problem on the execution host, as there will never be more then 10,000 tasks on one host, but the same spooling functions are used with classic spooling in the Qmaster, too.

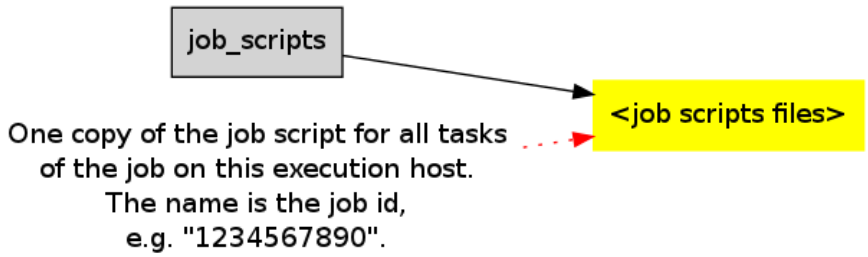


FIGURE 6: Overview of the Job Scripts Spooling Sub Directory

1.1.2 Displaying Status Information

Univa Grid Engine is a distributed system that handles and interacts with different entities like **jobs**, **hosts**, and **queues**.

- **queues** can have different states, depending on whether they are usable, non-usable, or they are in any special mode (like maintenance).
- With **jobs**, the states indicates things like if they are already started and when, if the jobs are running or if they are in any special state (like the suspended state).
- **Hosts** do not have an external state model, but they provide status information (like CPU or memory usage).

This section describes how the states and the status for the different objects can be displayed and how they are interpreted.

Displaying Job Status Information

After submitting a job, Univa Grid Engine handles the complete lifetime of the job and expresses the condition of the job in various predefined job states. A job can have multiple combined states, hence the total number of different job states is very high. Use the `qstat` command to show job states:

```
> qstat
job-ID prior name    user   state    submit/start at     queue     slots ja-task-ID
-----------------------------------------------------------------------------------
13 0.50500 sleep    daniel    r     05/24/2011 09:57:07 all.q@host2      1
14 0.50500 sleep    daniel    r     05/24/2011 09:57:07 all.q@host3      1
15 0.50500 sleep    daniel    r     05/24/2011 09:57:07 all.q@host1      1 1
15 0.50500 sleep    daniel    r     05/24/2011 09:57:07 all.q@host1      1 2
15 0.50500 sleep    daniel    r     05/24/2011 09:57:07 all.q@host1      1 3
15 0.50500 sleep    daniel    r     05/24/2011 09:57:07 all.q@host1      1 4
15 0.50500 sleep    daniel    r     05/24/2011 09:57:07 all.q@host1      1 5
15 0.50500 sleep    daniel    r     05/24/2011 09:57:07 all.q@host1      1 6
12 0.60500 env      daniel    qw    05/24/2011 09:56:45                  1
```

The job state is displayed in the **state** column. In this example, there are several jobs running (`r`) and one job is pending (queue waiting, `qw`).

Other basic status information are the queue instance in which the job is running (`queue`), the submit time (if the job is in the queued state) and the start time (if the job was dispatched already to queue instances).

Understanding the Various Job States Univa Grid Engine job states can be a combination of different states. For example, there are different hold states that can be applied to jobs during submit time or afterwards, when they are running. A **hold state** prevents a job from being considered during a scheduling run, therefore it affects a running job only when it is rescheduled.

The following example illustrates the hold state in combination with other states:

Here a job is submitted with a user hold (`-h`):

```
> qsub -h -b y sleep 120
Your job 16 ("sleep") has been submitted
```

After submission, the job stays in the combined hold queued-waiting state.

```
> qstat
job-ID  prior   name    user   state  submit/start at     queue  slots ja-task-ID
-----------------------------------------------------------------------------------
16     0.00000  sleep  daniel   hqw   05/24/2011 10:33:17               1
```

If the hold is removed and the job was dispatched, it is in the running state. When then a user hold for the job is requested, the `qstat` command shows the combined state **hold running**.

```
> qrls 16

modified hold of job 16

> qstat
job-ID prior   name     user   state  submit/start at     queue  slots ja-task-ID
-----------------------------------------------------------------------------------
16   0.00000  sleep  daniel    qw   05/24/2011 10:33:17               1

> qstat
job-ID  prior   name    user  state  submit/start at     queue     slots ja-task-ID
-----------------------------------------------------------------------------------
16    0.55500  sleep  daniel   r  05/24/2011 10:34:37 all.q@SLES11SP1      1

> qhold 16

modified hold of job 16

> qstat
job-ID  prior   name   user  state submit/start at     queue   slots ja-task-ID
-----------------------------------------------------------------------------------
16    0.55500  sleep  daniel  hr  05/24/2011 10:34:37 all.q@SLES11SP1      1
```

The following table provides an overview of the Univa Grid Engine job states, which can occur alone or in combination with other states:

State	Description
r	Running state. The job is running on the execution host.
t	Job is in a transferring state. The job is sent to the execution host.
d	The job is in a deletion state. The job is currently deleted by the system.
E	The job is in an error state.
R	The job was restarted.
T	The job is in a suspended state because of threshold limitations.
w	The job is in a waiting state.
h	The job is in a hold state. The hold state prevents scheduling of the job.
S	The job is in an automatic suspended state. The job suspension was triggered not directly.
s	The job is in a manual suspend state. The job suspension was triggered manually.
z	The job is in a zombie state.

Table 5: Overview of Job States

The graphic below illustrates a simple but common job state transition from queued-waiting (qw), to transferring (t) and running (r). While running, the job switches to the suspended state (s) and back.

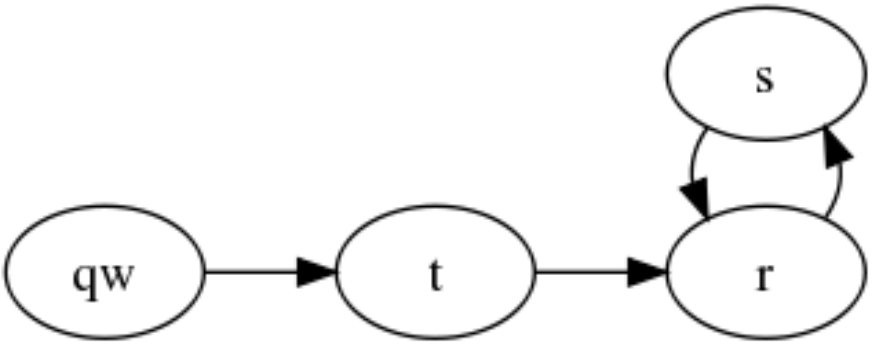


FIGURE 4: Simple Job State Transition

Displaying Host Status Information

Status information of hosts can be displayed with the `qhost` command. Hosts themselves have no pre-defined explicit states like jobs or queues. Depending on the internal host state, the **queue instances** on that host change its state. When for example a host is not reachable anymore, then all queue instances on that host go into the alarm state (`a`). Nevertheless status information and host topology based information remain available. Examples of status information are the architecture, the compute load, memory state. Examples of host topology based information are the number of sockets, cores and hardware supported threads (the latter on Linux and Solaris only).

Use the `qhost` command to show host status information, as in the following example:

```
> qhost
HOSTNAME     ARCH         NCPU NSOC NCOR NTHR  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------------
global          -           -    -    -    -     -       -       -       -       -
tanqueray    lx-amd64         2    1    2    2  0.27    7.7G    2.2G     0.0     0.0
unertl       lx-amd64         1    1    1    1  0.00  997.5M  299.0M    2.0G     0.0
```

Note

In order to get SGE 6.2u5 compatible output (without NSOC, NCOR, NTHR), use the `-ncb` switch (e.g. `qhost -ncb`).

More detailed host status information can be shown with the `-F` argument. In addition to the default `qhost` information, host-specific values (`hl:`) are also shown.

```
> qhost -F
HOSTNAME          ARCH      NCPU NSOC NCOR NTHR  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------------
global             -           -    -    -    -     -       -       -       -       -
host1              lx-amd64    8    1    8    8  0.00  491.9M   51.9M  398.0M     0.0
  hl:arch=lx-amd64
  hl:num_proc=8.000000
  hl:mem_total=491.898M
  hl:swap_total=397.996M
  hl:virtual_total=889.895M
  hl:load_avg=0.000000
  hl:load_short=0.000000
  hl:load_medium=0.000000
  hl:load_long=0.000000
  hl:mem_free=439.961M
  hl:swap_free=397.996M
  hl:virtual_free=837.957M
  hl:mem_used=51.938M
  hl:swap_used=0.000
  hl:virtual_used=51.938M
  hl:cpu=0.000000
  hl:m_topology=SCCCCCCCC
  hl:m_topology_inuse=SCCCCCCCC
  hl:m_socket=1.000000
  hl:m_core=8.000000
  hl:m_thread=8.000000
  hl:np_load_avg=0.000000
  hl:np_load_short=0.000000
  hl:np_load_medium=0.000000
  hl:np_load_long=0.000000
```

Understanding the Various Host States

The following descriptions refer to the column headers output by the `qhost` command:

```
> qhost
 HOSTNAME    ARCH     NCPU NSOC NCOR NTHR  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
 ----------------------------------------------------------------------------------
```

- `HOSTNAME`: The names of the available hosts.
- `ARCH`: The host architecture shown by the `qhost` command is either an abbreviation of the operating system used on the execution host (e.g. `aix51`) or a combination of the operating system and the processor architecture (e.g. `sol-amd64`).
- `NCPU`: The number of CPUs for a system is determined by an operating system call. In most cases, it is the number of available **CPU cores** on a system.
- The next three entries are **execution host topology** related information and are only available on Linux hosts (with a kernel version >= 2.6.16) and Solaris hosts.
  - `NSOC`: number of CPU sockets on the execution host
  - `NCOR`: total number of compute cores on the execution host
  - `NTHR`: hardware supported threads on the execution host
- `LOAD`: The **machine load** is the average length of the operating system run-queue (runnable processes) in the last 5 minutes (on some operating systems, this may differ). The source is the load value `load_avg`.
- The current **memory status** is displayed in the `MEMTOT` and `MEMUSE` columns.
  - `MEMTOT`: total amount of memory
  - `MEMUSE`: used memory
- **Virtual memory** specific information is shown in the `SWAPTO` and `SWAPUS` columns.
  - `SWAPTO`: total amount of swap space
  - `SWAPUS`: used swap space

Note

Your own host based load values can be added by declaring the load value name and typing in the complex configuration (`qconf -mc`) and initializing the load value either in the execution host configuration (`qconf -me <hostname>`) or by installing a `load sensor` at the execution host.

The following table explains these additional standard load values.

State	Description
arch	The architecture string (usually contains the OS and optionally the ISA).
num_proc	The number of detected processing units.
mem_total	The total amount of installed memory.

State	Description
swap_total	The total amount of installed swap space.
virtual_total	Total amount of virtual memory (memory + swap space).
load_avg	Same as load_medium.
load_short	Average load value in the last minute (time interval may differ on OS; source on Linux is `/proc/loadavg`).
load_medium	Average load value in the last 5 minutes (time interval may differ on OS; source on Linux is `/proc/loadavg`).
load_long	Average load value in the last 15 minutes (time interval may differ on OS; source on Linux is `/proc/loadavg`).
mem_free	The amount of unused memory.
swap_free	The amount of unused swap space.
virtual_free	The amount of unused virtual memory.
mem_used	The amount of occupied memory.
swap_used	The amount of occupied swap space.
virtual_used	The amount of occupied virtual memory.
cpu	Current amount of CPU usage.
m_topology	Execution host topology information (S means socket, C core, and T hardware supported thread).
m_topology_inuse	Execution host topology like above. Additionally occupied (via core binding) cores are displayed in lower case letters.
m_socket	The number of CPU sockets.
m_core	The total number of CPU cores.
m_thread	The total number of hardware supported threads.
np_load_avg	Medium average divided by number of processors (num_proc).
np_load_short	Short load average divided by the number of processors (num_proc).
np_load_medium	Medium load average divided by the number of processors (num_proc).
np_load_long	Long load average divided by the number of processors (num_proc).
display_win_gui	On Windows (win-x86) only, this value denotes if the execution host is able to display the GUI of a job on the currently visible desktop.

Table 6: Additional Standard Load Values

Displaying Queue Status Information

The `qstat` command shows queue status information.

Use the queue selection switch `-q` to show all queue instances of the `all.q`.

```
> qstat -q all.q -f
queuename                qtype resv/used/tot. load_avg      arch    states
 ---------------------------------------------------------------------------
 all.q@host1              BIPC  0/0/10          0.00      lx-amd64
 ---------------------------------------------------------------------------
 all.q@host2              BIPC  0/0/10          0.08      lx-amd64
 ---------------------------------------------------------------------------
 all.q@host3              BIPC  0/0/10          0.01      lx-amd64
```

Understanding the Various Queue States

The following table shows the different queue states, which can also occur in combination.

State	Description
a	Alarm state (because of load threshold or also when host is not reachable)
A	Alarm state
u	Unknown state: The execution daemon is not reachable.
C	Calendar suspended
s	Suspended
S	Automatically suspended
d	Manually disabled (`qmod -d`)
D	Automatically disabled
E	Error state

Table 7: Queue States

1.2 Understanding a Default Installation

These sections describe common parts of a default Univa Grid Engine installation. Topics covered include the queue, parallel environment, user sets, host groups and complex attributes.

1.2.1 Default Queue

All hosts are by default members of the queue **all.q**, where every installed execution host has as many slots as the number of CPUs reported by the operating system. This *default-queue* is configured to run batch, interactive and also parallel jobs with the C-Shell as default.

See Configuring Queues for more information, such as how to change the queue.

```
# qconf -sq all.q
qname                 all.q
hostlist              @allhosts
seq_no                0
```

```
load_thresholds       np_load_avg=1.75
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
rerun                 FALSE
slots                 1,[host1=4],[host2=1],[host3=1],[host4=1]
tmpdir                /tmp
shell                 /bin/sh
prolog                NONE
epilog                NONE
shell_start_mode      posix_compliant
starter_method        NONE
suspend_method        NONE
resume_method         NONE
terminate_method      NONE
notify                00:00:60
owner_list            NONE
user_lists            NONE
xuser_lists           NONE
subordinate_list      NONE
complex_values        NONE
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                INFINITY
h_core                INFINITY
s_rss                 INFINITY
h_rss                 INFINITY
s_vmem                INFINITY
h_vmem                INFINITY
```

1.2.2 Default PE

There is also a pre-defined parallel-environment configured named **make** which is also already added to the default queue. This pe utilizes at most 999 slots and allocates them with *round_robin* as the allocation rule.

See User Guide -> Parallel environments for more information on how to handle those parallel environments.

```
# qconf -sp make
 pe_name                make
 slots                  999
 used_slots             0
 bound_slots            0
 user_lists             NONE
 xuser_lists            NONE
 start_proc_args        NONE
 stop_proc_args         NONE
 per_pe_task_prolog     NONE
 per_pe_task_epilog     NONE
 allocation_rule        $round_robin
 control_slaves         TRUE
 job_is_first_task      FALSE
 urgency_slots          min
 accounting_summary     TRUE
 daemon_forks_slaves    FALSE
 master_forks_slaves    FALSE
```

1.2.3 Default User Set Lists

By default, there are three different user set lists defined. **arusers** and **deadlineusers** are *access lists* and **defaultdepartment** is a *department*.

All members of the *arusers* user set list and also the Univa Grid Engine operators and managers are allowed to do *advance reservations* (User Guide -> Reservations).

All members of the *deadlineusers* user set list and also the Univa Grid Engine operators and managers are allowed to submit *deadline jobs*.

See Managing User Access for more information.

```
 # qconf -sul
 arusers
 deadlineusers
 defaultdepartment
```

1.2.4 Default Host Group List

@allhosts is the only pre-defined host group list. All hosts known at install time of the Qmaster will be members of this host group list.

```
# qconf -shgrpl
 @allhosts
```

1.2.5 Default Complex Attributes

Many pre-defined “complex attributes” are available.

See [Configuring Complexes][Configuring Complexes] for additional information.

```
# qconf -sc
#name               shortcut   type        relop requestable consumable default  urgency aapre
#-------------------------------------------------------------------------------------------
arch                a          RESTRING    ##    YES         NO         NONE     0       NO
calendar            c          RESTRING    ##    YES         NO         NONE     0       NO
cpu                 cpu        DOUBLE      >=    YES         NO         0        0       NO
display_win_gui     dwg        BOOL        ##    YES         NO         0        0       NO
docker              dock       BOOL        ==    YES         NO         0        0       NO
docker_images       dockimg    RESTRING    ==    YES         NO         NONE     0       NO
h_core              h_core     MEMORY      <=    YES         NO         0        0       NO
h_cpu               h_cpu      TIME        <=    YES         NO         0:0:0    0       NO
h_data              h_data     MEMORY      <=    YES         NO         0        0       NO
h_fsize             h_fsize    MEMORY      <=    YES         NO         0        0       NO
h_rss               h_rss      MEMORY      <=    YES         NO         0        0       NO
h_rt                h_rt       TIME        <=    YES         NO         0:0:0    0       NO
h_stack             h_stack    MEMORY      <=    YES         NO         0        0       NO
h_vmem              h_vmem     MEMORY      <=    YES         NO         0        0       NO
hostname            h          HOST        ##    YES         NO         NONE     0       NO
load_avg            la         DOUBLE      >=    NO          NO         0        0       NO
load_long           ll         DOUBLE      >=    NO          NO         0        0       NO
load_medium         lm         DOUBLE      >=    NO          NO         0        0       NO
load_short          ls         DOUBLE      >=    NO          NO         0        0       NO
m_core              core       INT         <=    YES         NO         0        0       NO
m_socket            socket     INT         <=    YES         NO         0        0       NO
m_thread            thread     INT         <=    YES         NO         0        0       NO
m_topology          topo       RESTRING    ##    YES         NO         NONE     0       NO
m_topology_inuse    utopo      RESTRING    ##    YES         NO         NONE     0       NO
mem_free            mf         MEMORY      <=    YES         NO         0        0       NO
mem_total           mt         MEMORY      <=    YES         NO         0        0       NO
mem_used            mu         MEMORY      >=    YES         NO         0        0       NO
min_cpu_interval    mci        TIME        <=    NO          NO         0:0:0    0       NO
np_load_avg         nla        DOUBLE      >=    NO          NO         0        0       NO
np_load_long        nll        DOUBLE      >=    NO          NO         0        0       NO
np_load_medium      nlm        DOUBLE      >=    NO          NO         0        0       NO
np_load_short       nls        DOUBLE      >=    NO          NO         0        0       NO
num_proc            p          INT         ##    YES         NO         0        0       NO
qname               q          RESTRING    ##    YES         NO         NONE     0       NO
rerun               re         BOOL        ##    NO          NO         0        0       NO
s_core              s_core     MEMORY      <=    YES         NO         0        0       NO
s_cpu               s_cpu      TIME        <=    YES         NO         0:0:0    0       NO
```

```
s_data          s_data      MEMORY      <=    YES         NO          0         0        NO
s_fsize         s_fsize     MEMORY      <=    YES         NO          0         0        NO
s_rss           s_rss       MEMORY      <=    YES         NO          0         0        NO
s_rt            s_rt        TIME        <=    YES         NO          0:0:0     0        NO
s_stack         s_stack     MEMORY      <=    YES         NO          0         0        NO
s_vmem          s_vmem      MEMORY      <=    YES         NO          0         0        NO
seq_no          seq         INT         ##    NO          NO          0         0        NO
slots           s           INT         <=    YES         YES         1         1000     YES
swap_free       sf          MEMORY      <=    YES         NO          0         0        NO
swap_rate       sr          MEMORY      >=    YES         NO          0         0        NO
swap_rsvd       srsv        MEMORY      >=    YES         NO          0         0        NO
swap_total      st          MEMORY      <=    YES         NO          0         0        NO
swap_used       su          MEMORY      >=    YES         NO          0         0        NO
tmpdir          tmp         RESTRING    ##    NO          NO          NONE      0        NO
virtual_free    vf          MEMORY      <=    YES         NO          0         0        NO
virtual_total   vt          MEMORY      <=    YES         NO          0         0        NO
virtual_used    vu          MEMORY      >=    YES         NO          0         0        NO
```

1.3 Understanding Key Univa Grid Engine Configuration Objects

There are four key configuration objects that define the outline of a Univa Grid Engine cluster.

- cluster configuration
- scheduler configuration
- host configurations
- queues

Some of them are created and initialized during the installation process and some of them have to be created after the installation to setup necessary policies in the cluster.

1.3.1 The Cluster Configuration

The cluster configuration is a configuration object that defines global aspects of the cluster setup. Modification of this object requires manager privileges.

Certain settings of the global cluster configuration can be specified differently for individual submit and execution hosts in a cluster. For these hosts a local configuration object can be created. The local configuration object defines the parameters that should deviate from the global configuration.

The available parameters of the global and local configuration can be found in the chapter Understanding and Modifying the Cluster Configuration.

1.3.2 The Scheduler Configuration

All parameters influencing the scheduler component of Univa Grid Engine are summarized in the scheduler configuration. Only managers of a Univa Grid Engine cluster are allowed to change scheduler settings.

Scheduler configuration parameters are explained in in the chapter Understanding and Modifying the Univa Grid Engine Scheduler Configuration.

1.3.3 Host and Queue Configurations

Hosts and cluster queues define the execution environment where jobs will be executed. The host configuration object defines aspects of an execution host. Cluster queues are used to partition a group of hosts and to provide more detailed setting that jobs require to get executed properly.

Read the section Configuration Properties of Hosts and Queues to get more information how to setup and configure those objects.

1.4 Navigating the ARCo Database

1.4.1 Accessing the ARCo Database

Use a database frontend to access the ARCo database, e.g. a reporting tool, or spreadsheet.

During dbwriter installation, a user *arco_read* has been created having read access to the ARCo database. This *arco_read* user should be used to connect a reporting tool to the ARCo database.

The examples in the following sections use a PostgreSQL database and the `psql` command line tool.

1.4.2 Views to the Database

To make querying data from the ARCo database easier, a number of views have been created on the ARCo tables.

It is recommended to use these views where possible.

The following views are available:

- Accounting
  - view_accounting: Accounting information per job.
- Job related information
  - view_job_log: The job logging.
  - view_job_times: Timing information for jobs.
  - view_jobs_completed: Number of jobs completed over time.
- Advance reservation data
  - view_ar_attribute: Attributes of advance reservations.
  - view_ar_log: The AR log (state changes of an AR).
  - view_ar_resource_usage: Resources requested by advance reservations.
  - view_ar_time_usage: Reserved time vs. time actually used by slots.
  - view_ar_usage: Timing information of advance reservations.

- Values related to Univa Grid Engine configuration objects
  - view_department_values: Values related to departments.
  - view_group_values: Values related to user groups.
  - view_host_values: Values related to hosts.
  - view_project_values: Values related to projects.
  - view_queue_values: Values related to queue instances.
  - view_user_values: Values related to users.
- Statistics
  - view_statistic: ARCo statistics.

The following detailed view documentation is based on a PostgreSQL database. The database structure is the same for all supported database systems, but with the attribute types there are slight differences.

Accounting view_accounting

The view_accounting gives basic accounting information for finished jobs. More detailed information, e.g. the rusage (ru_*) attributes can be retrieved from the [sge_job_usage table][sge_job_usage].

Attribute	Type	Description
job_number	bigint	the job id
task_number	bigint	the array task id
pe_taskid	text	the id of a task of a tightly integrated parallel job
name	text	the job name
group	text	the user group of the job owner (submitter)
username	text	the user name of the job owner (submitter)
account	text	the account string (see qsub -A option)
project	text	the project the job belongs to
department	text	the department the job owner belongs to
submission_time	timestamp without time zone	the time when the job was submitted
ar_parent	numeric(38,0)	a reference to the advance reservation the job is running in, see [sge_ar table][sge_ar].
start_time	timestamp without time zone	the time when the job (the array task) was started
end_time	timestamp without time zone	the time when the job (the array task) finished
wallclock_time	double precision	the job run time in seconds

Attribute	Type	Description
cpu	double precision	the cpu time consumed in seconds
mem	double precision	the integral memory usage in GB seconds
io	double precision	the amount of data transferred in input/output operations (available only on certain architectures)
iow	double precision	the io wait time in seconds (available only on certain architectures)
maxvmem	double precision	the maximum vmem size in bytes
exit_status	integer	the exit status of the job

Table 8: Information Available from `view_accounting`

See also the man page accounting.5 for more information.

Example:

How many job have been run and how much cpu time has been consumed during the last hour, listed per user:

```
SELECT username, count(*) AS jobs, sum(cpu)
   FROM view_accounting
   WHERE end_time > date_trunc('hour', now())
   GROUP BY username
   ORDER BY username;

 username | jobs |    cpu
----------+------+-----------
 joga     |  175 | 10.612507
 sgetest  |  181 |  4.792978
 sgetest1 |  186 |  4.956504
 sgetest2 |  276 |  7.054217
```

Job related information

In addition to the [Accounting][Accounting], there are views showing more details of jobs, like the job log, job timing information and a summary about finished jobs.

view_job_log

The job log shows detailed status information about the whole life cycle of a job, from job submission to the job end.

Attribute	Type	Description
job_number	bigint	the job id

Attribute	Type	Description
task_number	bigint	the array task id
pe_taskid	text	the id of a task of a tightly integrated parallel job
name	text	the job name
group	text	the user group of the job owner
username	text	the name of the job owner
account	text	the account string (see qsub -A option)
project	text	the project the job was running in
department	text	the department the job owner belongs to
time	timestamp without time zone	the time when a job log event occurred
event	text	name of the job log event (e.g. pending, delivered, finished)
state	text	the job state (e.g. r for running)
initiator	text	the initiator of the event, e.g. the name of the operator who suspended the job
host	text	the host from which the event was triggered
message	text	a message describing the event

Table 9: Information Available from the Job Log

Example:

```
SELECT job_number, time, event, state, initiator, message
   FROM view_job_log
   WHERE job_number = 59708
   ORDER BY time;
```

```
 job_number  |   time     |    event    | state |  initiator  |    message
-------------+------------+-------------+-------+-------------+-------------------
 59708       | 2011-05-24 | pending     |       | joga        | new job
             | 12:02:17   |             |       |             |
 59708       | 2011-05-24 | sent        | t     | master      | sent to execd
             | 12:02:35   |             |       |             |
 59708       | 2011-05-24 | delivered   | r     | master      | job received by execd
             | 12:02:36   |             |       |             |
 59708       | 2011-05-24 | suspended   | r     | joga        |
             | 12:02:44   |             |       |             |
 59708       | 2011-05-24 | unsuspended | r     | joga        |
             | 12:03:01   |             |       |             |
```

```
59708       | 2011-05-24 | finished    | r     | execution   | job exited
            | 12:03:35   |             |       |  daemon     |
59708       | 2011-05-24 | finished    | r     | master      | job waits for schedds
            | 12:03:35   |             |       |             | deletion
59708       | 2011-05-24 | deleted     | T     | scheduler   | job deleted by schedd
            | 12:03:35   |             |       |             |
```

view_job_times

The view_job_times gives timing information about a job, like when a job was submitted, started, finished as well as the job run time, the total turnaround time, and so on.

Attribute	Type	Description
job_number	bigint	the job id
task_number	bigint	the array task id, -1 for non array jobs
name	text	the job name
groupname	text	the user group of the job owner
username	text	the user name of the job owner
account	text	the account string (see qsub -A option)
project	text	the project the job was belonging to
department	text	the department the job owner belongs to
submission_time	timestamp without time zone	the job submission time
ar_parent	numeric(38,0)	reference to an advance reservation the job was running in
start_time	timestamp without time zone	the time when the job was started
end_time	timestamp without time zone	the time when the job finished
wait_time	interval	the time between job submission and job start as time interval (e.g. 00:00:10)
turnaround_time	interval	the total job turnaround time (time from job submission until job end as time interval)
job_duration	interval	the job run time as time interval
wallclock_time	integer	the job run time in seconds
exit_status	integer	the exit status of the job

Table 10: Information Available from `view_job_times`

Example: Look for jobs that were pending more than 3 minutes before getting scheduled:

```
SELECT job_number, task_number, submission_time, wait_time, start_time, end_time
```

```
        FROM view_job_times
        WHERE wait_time > '00:03:00'
        ORDER BY wait_time;

job_  | task_  |  submission_time    |wait_time |     start_time      | end_time
number| number |                     |          |                     |
------+--------+---------------------+----------+---------------------+--------------
4732  |     34 | 2011-05-23 14:32:43 | 00:03:07 | 2011-05-23 14:35:50 | 2011-05-23 14:36:00
4695  |     -1 | 2011-05-23 14:28:49 | 00:03:08 | 2011-05-23 14:31:57 | 2011-05-23 14:32:12
4732  |     35 | 2011-05-23 14:32:43 | 00:03:09 | 2011-05-23 14:35:52 | 2011-05-23 14:36:02
4732  |     36 | 2011-05-23 14:32:43 | 00:03:17 | 2011-05-23 14:36:00 | 2011-05-23 14:36:10
4732  |     37 | 2011-05-23 14:32:43 | 00:03:20 | 2011-05-23 14:36:03 | 2011-05-23 14:36:13
4732  |     38 | 2011-05-23 14:32:43 | 00:03:28 | 2011-05-23 14:36:11 | 2011-05-23 14:36:21
```

view_jobs_completed

The view_jobs_completed shows the number of jobs finished per hour.

Attribute	Type	Description
time	timestamp without time zone	start time of a time interval
completed	bigint	number of jobs completed between time and time + 1 hour
ar_parent	numeric(38,0)	if advance reservations are used, the completed jobs are listed per time interval and advance reservation

Table 11: Information Available from `view_jobs_completed`

Example: Show number of jobs that completed during the last 24 hours, summed up per hour:

```
    SELECT *
        FROM view_jobs_completed
        WHERE time > date_trunc('day', now());

 time                 | completed | ar_parent
 ---------------------+-----------+-----------
  2011-05-24 01:00:00 |      2712 |         0
  2011-05-24 02:00:00 |      2715 |         0
  2011-05-24 03:00:00 |      2713 |         0
  2011-05-24 04:00:00 |      2712 |         0
  2011-05-24 05:00:00 |      2715 |         0
  2011-05-24 06:00:00 |      2712 |         0
  2011-05-24 07:00:00 |      2714 |         0
  2011-05-24 08:00:00 |      2713 |         0
  2011-05-24 09:00:00 |      2178 |         0
```

```
2011-05-24 10:00:00 |      1574 |         0
2011-05-24 10:00:00 |         3 |         1
2011-05-24 11:00:00 |      1109 |         0
2011-05-24 12:00:00 |       201 |         0
```

Advance Reservation Data

view_ar_attribute

The view_ar_attribute shows the basic attributes of an advance reservation.

Attribute	Type	Description
ar_number	bigint	the ar number
owner	text	the owner of the advance reservation
submission_time	timestamp without time zone	the time when the ar was submitted
name	text	the name of the ar
account	text	the account string (see qrsub -A option)
start_time	timestamp without time zone	the start time of the advance reservation
end_time	timestamp without time zone	the end time of the advance reservation
granted_pe	text	name of a parallel environment which was granted to the advance reservation

Table 12: Information Available from `view_ar_attribute`

Example:

```
    SELECT * FROM view_ar_attribute;

ar_    | owner | submission_  | name | account |  start_    |  end_     | granted_pe
number |       |       time   |      |         |    time    |  time     |
-------+-------+---------- ---+------+---------+---- -------+-----------+--------------
  1    | joga  | 2011-05-24   |      |   sge   | 2011-05-24 | 2011-05-24|
       |       |  09:59:48    |      |         |  10:30:00  | 11:30:00  |
```

view_ar_log

Attribute	Type	Description
ar_number	bigint	the ar number
time	timestamp without time zone	time when the event logged occurred
event	text	type of the event

Attribute	Type	Description
state	text	the state of the advance reservation
message	text	a message describing the event

Table 13: Information Available from `view_ar_log`

Example:

```
SELECT * FROM view_ar_log;

ar_    |        time         |         event          | state |    message
number |                     |                        |       |
-------+---------------------+------------------------+-------+---------------------
1      | 2011-05-24 09:59:48 | RESOURCES UNSATISFIED | W     | AR resources unsatisfied
1      | 2011-05-24 10:01:11 | RESOURCES SATISFIED   | w     | AR resources satisfied
1      | 2011-05-24 10:30:00 | START TIME REACHED    | r     | start time of AR reached
```

view_ar_resource_usage

Attribute	Type	Description
ar_number	bigint	the ar number
variable	text	name of a resource requested by the ar
value	text	requested value of the named resource

Table 14: Information Available from `view_ar_resource_usage`

Example:

```
SELECT * FROM view_ar_resource_usage;

ar_number  | variable |   value
-----------+----------+-----------
1          | arch     | sol-amd64
```

view_ar_time_usage

The view_ar_time shows the time resources were held by an advance reservation vs. the time these resources had actually been in use by jobs.

Attribute	Type	Description
ar_id	numeric(38,0)	the ar number

Attribute	Type	Description
job_duration	interval	actual usage of the reserved resources by jobs
ar_duration	interval	duration (time interval) of the advance reservation

Table 15: Information Available from `view_ar_time_usage`

Example:

```
    SELECT * FROM view_ar_time_usage;

 ar_id  | job_duration  | ar_duration
--------+---------------+-------------
 1      | 00:03:00      | 01:00:00
```

view_ar_usage

The view_ar_usage shows until when which queue instances (cluster queue on host) had been in use by an advance reservation.

Attribute	Type	Description
ar_number	bigint	the ar number
termination_time	timestamp without time zone	time when the ar finished
queue	text	cluster queue name
hostname	text	host name
slots	integer	number of slots reserved on the named queue instance

Table 16: Information Available from `view_ar_usage`

Example:

```
   SELECT * FROM view_ar_usage;

 ar_number  |  termination_time   | queue | hostname | slots
------------+---------------------+-------+----------+-------
  1         | 2011-05-24 11:30:00 | all.q | hookipa  |     1
```

Values Related to Univa Grid Engine Configuration Objects

Arbitrary values can be stored in the ARCo database related to the following Univa Grid Engine configuration objects:

- departments
- user groups
- hosts
- projects
- queues
- users

Examples for values related to such objects are

- load values of hosts
- license counters
- number of jobs completed per
  - department
  - user
  - project
- configured vs. free queue slots
- ... and more.

Object related values are valid for a certain time period, meaning they have a start and an end time.

A number of views allows easy access to these values.

view_department_values

Attribute	Type	Description
department	text	name of the department
time_start	timestamp without time zone	value is valid from time_start on
time_end	timestamp without time zone	until time_end
variable	text	name of the variable
str_value	text	current value of the variable as string
num_value	double precision	current value of the variable as floating point number
num_config	double precision	configured capacity of the value (for consumables)

Table 17: Information Available from `view_department_values`

view_group_values

Attribute	Type	Description
groupname	text	name of the department
time_start	timestamp without time zone	value is valid from time_start on
time_end	timestamp without time zone	until time_end
variable	text	name of the variable
str_value	text	current value of the variable as string
num_value	double precision	current value of the variable as floating point number
num_config	double precision	configured capacity of the value (for consumables)

Table 18: Information Available from `view_group_values`

view_host_values

Attribute	Type	Description
hostname	text	name of the host
time_start	timestamp without time zone	value is valid from time_start on
time_end	timestamp without time zone	until time_end
variable	text	name of the variable
str_value	text	current value of the variable as string
num_value	double precision	current value of the variable as floating point number
num_config	double precision	configured capacity of the value (for consumables)

Table 19: Information Available from `view_host_values`

Example: Show the average load per hour during the last day:

```
SELECT hostname, variable, time_end, num_value
   FROM view_host_values
   WHERE variable = 'h_load' AND time_end > date_trunc('day', now())
   ORDER BY time_end, hostname;
```

```
 hostname | variable |      time_end       |      num_value
----------+----------+---------------------+----------------------
 halape   | h_load   | 2011-05-25 01:00:00 | 0.000465116279069767
 hapuna   | h_load   | 2011-05-25 01:00:00 |   0.0108707865168539
 hookipa  | h_load   | 2011-05-25 01:00:00 |   0.0738077368421051
 kahuku   | h_load   | 2011-05-25 01:00:00 |   0.0430645161290322
 kailua   | h_load   | 2011-05-25 01:00:00 |  0.00572881355932204
 kehena   | h_load   | 2011-05-25 01:00:00 | 0.000635838150289017
 rgbfs    | h_load   | 2011-05-25 01:00:00 |             0.092773
 rgbtest  | h_load   | 2011-05-25 01:00:00 |   0.0061759138888889
 halape   | h_load   | 2011-05-25 02:00:00 |                    0
 hapuna   | h_load   | 2011-05-25 02:00:00 |   0.0101123595505618
...
```

view_project_values

Attribute	Type	Description
project	text	name of the project
time_start	timestamp without time zone	value is valid from time_start on
time_end	timestamp without time zone	until time_end
variable	text	name of the variable
str_value	text	current value of the variable as string
num_value	double precision	current value of the variable as floating point number
num_config	double precision	configured capacity of the value (for consumables)

Table 20: Information Available from `view_project_values`

view_queue_values

A queue value is related to a queue instance (cluster queue on a specific host).

Attribute	Type	Description
qname	text	name of the cluster queue
hostname	text	name of the host
time_start	timestamp without time zone	value is valid from time_start on
time_end	timestamp without time zone	until time_end
variable	text	name of the variable

Attribute	Type	Description
str_value	text	current value of the variable as string
num_value	double precision	current value of the variable as floating point number
num_config	double precision	configured capacity of the value (for consumables)

Table 21: Information Available from `view_queue_values`

Example: Show the configured capacity for slots and the actually used slots per queue instance over the last hour:

```
SELECT qname, hostname, time_end, variable, num_config, num_value
   FROM view_queue_values
   WHERE variable = 'slots' AND time_end > date_trunc('hour', now())
   ORDER BY time_end, hostname;

 qname | hostname |      time_end       | variable | num_config | num_value
-------+----------+---------------------+----------+------------+-----------
 all.q | rgbfs    | 2011-05-25 08:00:01 | slots    |         40 |         0
 all.q | rgbtest  | 2011-05-25 08:00:01 | slots    |         60 |         3
 all.q | hapuna   | 2011-05-25 08:00:05 | slots    |         10 |         0
 all.q | rgbtest  | 2011-05-25 08:00:05 | slots    |         60 |         4
 all.q | hapuna   | 2011-05-25 08:00:13 | slots    |         10 |         1
 all.q | rgbtest  | 2011-05-25 08:00:14 | slots    |         60 |         3
 all.q | rgbtest  | 2011-05-25 08:00:16 | slots    |         60 |         3
 all.q | rgbtest  | 2011-05-25 08:00:16 | slots    |         60 |         4
 all.q | rgbtest  | 2011-05-25 08:00:19 | slots    |         60 |         3
 all.q | rgbtest  | 2011-05-25 08:00:19 | slots    |         60 |         4
```

view_user_values

Attribute	Type	Description
username	text	name of the user
time_start	timestamp without time zone	value is valid from time_start on
time_end	timestamp without time zone	until time_end
variable	text	name of the variable
str_value	text	current value of the variable as string
num_value	double precision	current value of the variable as floating point number

Attribute	Type	Description
num_config	double precision	configured capacity of the value (for consumables)

Table 22: Information Available from `view_user_values`

Example: Show the number of slots finished per hour and user for the last day:

```
SELECT username, time_end, num_value AS jobs_finished
  FROM view_user_values
  WHERE variable = 'h_jobs_finished' AND time_end > date_trunc('day', now())
  ORDER BY time_end, username;

  username |      time_end       | jobs_finished
 ----------+---------------------+---------------
  joga     | 2011-05-25 01:00:00 |           247
  sgetest  | 2011-05-25 01:00:00 |           245
  sgetest1 | 2011-05-25 01:00:00 |           246
  sgetest2 | 2011-05-25 01:00:00 |           245
  joga     | 2011-05-25 02:00:00 |           249
  sgetest  | 2011-05-25 02:00:00 |           246
  sgetest1 | 2011-05-25 02:00:00 |           245
  sgetest2 | 2011-05-25 02:00:00 |           245
```

Statistics

view_statistic

Shows statistical values.

A statistic has a name and can comprise multiple variables and their values over time.

Attribute	Type	Description
name	text	name of the statistic
time_start	timestamp without time zone	start time for validity of value
time_end	timestamp without time zone	end time for validity of value
variable	text	name of the variable
num_value	double precision	value of the variable

Table 23: Information Available from `view_statistic`

Example: Show the average processing speed of dbwriter per hour for the last day:

```
SELECT time\_end AS time, num\_value AS lines\_per\_second
   FROM view_statistic
   WHERE name = 'dbwriter' AND variable = 'h_lines_per_second'
      AND time_end > date_trunc('day', now())
   ORDER BY time_end;

   time                | lines_per_second
 ----------------------+------------------
  2011-05-25 01:00:00 | 3730.85662575603
  2011-05-25 02:00:00 |  3590.4583316432
  2011-05-25 03:00:00 | 3669.95984348156
  2011-05-25 04:00:00 | 3797.30899245708
  2011-05-25 05:00:00 | 3659.50091748412
  2011-05-25 06:00:00 | 3727.94193461027
  2011-05-25 07:00:00 |  3582.6273350896
  2011-05-25 08:00:00 | 3687.65701312245
```

Example: Show daily values for the number of records in the sge_host_values table:

```
SELECT * FROM view_statistic
   WHERE name = 'sge_host_values' AND variable = 'd_row_count';

   name           |    time_start       |      time_end       |  variable     | num_value
                  |                     |                     |               |
 -----------------+---------------------+---------------------+---------------+---------
                  |                     |                     |               |
  sge_host_values | 2011-05-23 00:00:00 | 2011-05-24 00:00:00 | d_row_count   |  82356
  sge_host_values | 2011-05-24 00:00:00 | 2011-05-25 00:00:00 | d_row_count   | 306459
```

1.4.3 Database Tables

The views described above are based on the raw data in the ARCo database tables.

The database tables have similar categories as the views:

- Job data and accounting
  - sge_job:
  - sge_job_log:
  - sge_job_request:
  - sge_job_usage:
- Advance reservation data
  - sge_ar:
  - sge_ar_attribute:
  - sge_ar_log:
  - sge_ar_resource_usage:

  - sge_ar_usage:

- Values related to Univa Grid Engine configuration objects

  - sge_department:
  - sge_department_values:
  - sge_group:
  - sge_group_values:
  - sge_host:
  - sge_host_values:
  - sge_project:
  - sge_project_values:
  - sge_queue:
  - sge_queue_values:
  - sge_user:
  - sge_user_values:

- Sharetree Usage

  - sge_share_log:

- Statistics

  - sge_statistic:
  - sge_statistic_values:

- dbwriter internal Data

  - sge_checkpoint:
  - sge_version:

The following detailed table documentation is based on a PostgreSQL database. The database structure is the same for all supported database systems, but with the attribute types there are slight differences.

Job Data and Accounting

sge_job

Attribute	Type	Description
j_id	numeric(38,0) not null	internal sequential id
j_job_number	bigint	the job number
j_task_number	bigint	the array task number, -1 for sequential jobs
j_pe_taskid	text	the id of tasks of tightly integrated parallel jobs, -1 for sequential jobs
j_job_name	text	the job name from qsub option -N
j_group	text	the name of the unix user group of the submit user

Attribute	Type	Description
j_owner	text	name of the submit user
j_account	text	account string from qsub option -A
j_priority	integer	the job priority set with qsub option -p
j_submission_time	timestamp without time zone	the job submission time
j_project	text	the project the job is submitted into (qsub option -P)
j_department	text	the department the job owner is assigned to
j_job_class	text	if the job was submitted into a job class, the name of the job class
j_submit_host	text	name of the submit host
j_submit_cmd	text	the command line with which the job was submitted

Table 24: Information Available from `sge_job`

sge_job_log

Attribute	Type	Description
jl_id	numeric(38,0) not null	
jl_parent	numeric(38,0)	reference to sge_job.j_id
jl_time	timestamp without time zone	time stamp of the job log event
jl_event	text	name of the job log event (e.g. pending, delivered, finished)
jl_state	text	the job state (e.g. r for running)
jl_user	text	the initiator of the event, e.g. the name of the operator who suspended the job
jl_host	text	the host from which the event was triggered
jl_state_time	integer	time stamp of the event generation, might be earlier than the jl_time
jl_message	text	a message describing the event

Table 25: Information Available from `sge_job_log`

sge_job_request

Attribute	Type	Description
jr_id	numeric(38,0) not null	internal id
jr_parent	numeric(38,0)	reference to sge_job.j_id
jr_variable	text	name of a complex variable requested by the job
jr_value	text	requested value

Table 26: Information Available from `sge_job_request`

sge_job_usage

Holds job accounting data.

See also man page accounting.5 for details.

Attribute	Type	Description
ju_id	numeric(38,0) not null	internal sequential id
ju_parent	numeric(38,0)	reference to sge_job.j_id
ju_curr_time	timestamp without time zone	time when the accounting record got requested
ju_qname	text	cluster queue name in which the job was running
ju_hostname	text	name of the host the job was running on
ju_start_time	timestamp without time zone	start time
ju_end_time	timestamp without time zone	end time
ju_failed	integer	indicates job start failures
ju_exit_status	integer	the job exit status
ju_granted_pe	text	name of the parallel environment in case of parallel jobs
ju_slots	integer	number of slots granted
ju_ru_wallclock	double precision	job run time
ju_ru_utime	double precision	user cpu time consumed
ju_ru_stime	double precision	system cpu time consumed
ju_ru_maxrss	integer	attributes delivered by the getrusage system call. Depending on the operating system only certain attributes are used. See man page getrusage.2 or getrusage.3c.
ju_ru_ixrss	integer	

Attribute	Type	Description
ju_ru_issmrss	integer	
ju_ru_idrss	integer	
ju_ru_isrss	integer	
ju_ru_minflt	integer	
ju_ru_majflt	integer	
ju_ru_nswap	integer	
ju_ru_inblock	integer	
ju_ru_outblock	integer	
ju_ru_msgsnd	integer	
ju_ru_msgrcv	integer	
ju_ru_nsignals	integer	
ju_ru_nvcsw	integer	
ju_ru_nivcsw	integer	
ju_cpu	double precision	the cpu time usage in seconds.
ju_mem	double precision	the integral memory usage in Gbytes cpu seconds.
ju_io	double precision	the amount of data transferred in input/output operations. Delivered only on certain operating systems.
ju_iow	double precision	the io wait time in seconds. Delivered only on certain operating systems.
ju_ioops	integer	the number of io operations. Delivered only on certain operating systems.
ju_maxvmem	double precision	the maximum vmem size in bytes.
ju_ar_parent	numeric(38,0) default 0	reference to sge_ar.ar_id
ju_qdel_info	text	information by whom a job was deleted
ju_maxrss	double precision	the maximum resident set size in bytes
ju_maxpss	double precision	the maximum proportional set size in bytes
ju_cwd	text	the jobs current working directory
ju_wallclock	double precision	the wallclock time measured by sge_execd for the job, excluding suspended times.

Table 27: Information Available from `sge_job_usage`

sge_job_online_usage

Holds job online usage data (usage information gathered during the job run time)

Attribute	Type	Description
jou_id	numeric(38,0) not null	internal sequential id
jou_parent	numeric(38,0) not null	reference to sge_job.j_id
jou_curr_time	timestamp	time when the online usage information got generated in sge_execd
jou_variable	text	name of the resource (e.g. cpu, maxvmem)
jou_dvalue	double precision	value of resource jou_variable at time jou_curr_time

Table 28: Information Available from `sge_job_online_usage`

Advance Reservation Data

sge_ar

Attribute	Type	Description
ar_id	numeric(38,0) not null	internal sequential id
ar_number	bigint	AR number
ar_owner	text	owner of the advance reservation
ar_submission_time	timestamp without time zone	submission time of the AR

Table 29: Information Available from `sge_ar`

sge_ar_attribute

Attribute	Type	Description
ara_id	numeric(38,0) not null	internal sequential id
ara_parent	numeric(38,0)	reference to sge_ar.ar_id
ara_curr_time	timestamp without time zone	time stamp of the AR reporting
ara_name	text	name of the AR
ara_account	text	account string from qrsub -A option
ara_start_time	timestamp without time zone	start time of the AR
ara_end_time	timestamp without time zone	end time of the AR
ara_granted_pe	text	if a parallel environment was requested at AR submission time, the name of the granted parallel environment

Attribute	Type	Description
ara_sr_cal_week	text	in case of standing reservation the week calendar describing the reservation points
ara_sr_depth	integer	in case of standing reservation the SR depth (the number of reservations being done at a time)
ara_sr_jmp	integer	in case of standing reservation the SR id (a number >= 0), in case of advance reservation -1

Table 30: Information Available from `sge_ar_attribute`

sge_ar_log

Attribute	Type	Description
arl_id	numeric(38,0) not null	internal sequential id
arl_parent	numeric(38,0)	reference to sge_ar.ar_id
arl_time	timestamp without time zone	time stamp of the ar log event
arl_event	text	type of the event
arl_state	text	the state of the advance reservation
arl_message	text	a message describing the event
arl_sr_id	bigint	in case of standing reservation the SR id (a number >= 0), in case of advance reservation -1

Table 31: Information Available from `sge_ar_log`

sge_ar_resource_usage

Attribute	Type	Description
arru_id	numeric(38,0) not null	internal sequential id
arru_parent	numeric(38,0)	reference to sge_ar.ar_id
arru_variable	text	name of a resource requested by the ar
arru_value	text	requested value of the named resource

Table 32: Information Available from `sge_ar_resource_usage`

sge_ar_usage

The sge_ar_usage table holds the information how many slots were reserved by an AR in which queue instance.

Attribute	Type	Description
aru_id	numeric(38,0) not null	internal sequential id
aru_parent	numeric(38,0)	reference to sge_ar.ar_id
aru_termination_time	timestamp without time zone	time when the reservation ended
aru_qname	text	cluster queue name
aru_hostname	text	host name
aru_slots	integer	number of slots reserved

Table 33: Information Available from `sge_ar_usage`

Values Related to Univa Grid Engine Configuration Objects

The Univa Grid Engine object related tables hold minimal information about the following configuration objects:

- departments
- user groups
- hosts
- projects
- queues
- users
- job classes

Records are inserted as soon as they are needed, e.g. when load values are stored into the ARCo database for an execution host, or when user related values are generated by derived value rules.

sge_department

The sge_department table contains one record per department configured in Univa Grid Engine.

Attribute	Type	Description
d_id	numeric(38,0) not null	internal sequential id
d_department	text	the department name

Table 34: Information Available from `sge_department`

sge_group

The sge_group table holds one record per Unix group name. New groups are generated as needed when jobs get submitted with a new group name.

Attribute	Type	Description
g_id	numeric(38,0) not null	internal sequential id
g_group	text	Unix group name

Table 35: Information Available from `sge_group`

sge_host

The sge_host table holds the names of all execution hosts.

Attribute	Type	Description
h_id	numeric(38,0) not null	internal sequential id
h_hostname	text	host name

Table 36: Information Available from `sge_host`

sge_project

The sge_project table holds project names.

Attribute	Type	Description
p_id	numeric(38,0) not null	internal sequential id
p_project	text	project name

Table 37: Information Available from `sge_project`

sge_queue

The sge_queue table holds one record per queue instance.

Attribute	Type	Description
q_id	numeric(38,0) not null	internal sequential id
q_qname	text	cluster queue name
q_hostname	text	host name

Table 38: Information Available from `sge_queue`

sge_user

Attribute	Type	Description
u_id	numeric(38,0) not null	internal sequential id
u_user	text	user name

Table 39: Information Available from `sge_user`

sge_job_class

Attribute	Type	Description
jc_id	numeric(38,0) not null	internal sequential id
jc_name	text	name of the job class

Table 40: Information Available from `sge_job_class`

For every Univa Grid Engine configuration object type stored in the ARCo database, there is also a table storing name/value pairs that hold data related to a configuration object, such as load values for execution hosts, consumable values for execution hosts or queues, values calculated for users or projects by derived value rules.

The value tables all store the following:

- timing information (start and end time for value validity)
- a variable name
- a configured capacity, only used for consumable resources
- the value of the variable during the reported time interval, can be a string value or a numeric value

sge_department_values

The sge_department_values table holds name/values pairs related to departments.

Information Available from `sge_department_values`

Attribute	Type	Description
dv_id	numeric(38,0) not null	internal sequential id
dv_parent	numeric(38,0)	reference to sge_department.d_id
dv_time_start	timestamp without time zone	time interval for the validity of the reported value
dv_time_end	timestamp without time zone	time interval for the validity of the reported value

Attribute	Type	Description
dv_variable	text	variable name
dv_svalue	text	variable value for string variables
dv_dvalue	double precision	variable value for numeric variables
dv_dconfig	double precision	configured capacity for consumables

Table 41: Information Available from `sge_department_values`

sge_group_values

The sge_group_values table holds name/values pairs related to Unix user groups.

Attribute	Type	Description
gv_id	numeric(38,0) not null	internal sequential id
gv_parent	numeric(38,0)	reference to sge_group.d_id
gv_time_start	timestamp without time zone	time interval for the validity of the reported value
gv_time_end	timestamp without time zone	time interval for the validity of the reported value
gv_variable	text	variable name
gv_svalue	text	variable value for string variables
gv_dvalue	double precision	variable value for numeric variables
gv_dconfig	double precision	configured capacity for consumables

Table 42: Information Available from `sge_group_values`

sge_host_values

The sge_host_values table holds name/values pairs related to execution hosts.

Attribute	Type	Description
hv_id	numeric(38,0) not null	internal sequential id
hv_parent	numeric(38,0)	reference to sge_host.d_id
hv_time_start	timestamp without time zone	time interval for the validity of the reported value
hv_time_end	timestamp without time zone	time interval for the validity of the reported value
hv_variable	text	variable name
hv_svalue	text	variable value for string variables

Attribute	Type	Description
hv_dvalue	double precision	variable value for numeric variables
hv_dconfig	double precision	configured capacity for consumables

Table 43: Information Available from `sge_host_values`

sge_project_values

The sge_project_values table holds name/values pairs related to projects.

Attribute	Type	Description
pv_id	numeric(38,0) not null	internal sequential id
pv_parent	numeric(38,0)	reference to sge_project.p_id
pv_time_start	timestamp without time zone	time interval for the validity of the reported value
pv_time_end	timestamp without time zone	time interval for the validity of the reported value
pv_variable	text	variable name
pv_svalue	text	variable value for string variables
pv_dvalue	double precision	variable value for numeric variables
dpv_dconfig	double precision	configured capacity for consumables

Table 44: Information Available from `sge_project_values`

sge_queue_values

The sge_queue_values table holds name/values pairs related to queue instances.

Attribute	Type	Description
qv_id	numeric(38,0) not null	internal sequential id
qv_parent	numeric(38,0)	reference to sge_queue.q_id
qv_time_start	timestamp without time zone	time interval for the validity of the reported value
qv_time_end	timestamp without time zone	time interval for the validity of the reported value
qv_variable	text	variable name
qv_svalue	text	variable value for string variables
qv_dvalue	double precision	variable value for numeric variables
qv_dconfig	double precision	configured capacity for consumables

Attribute	Type	Description

Table 45: Information Available from `sge_queue_values`

sge_user_values

The sge_user_values table holds name/values pairs related to user names.

Attribute	Type	Description
uv_id	numeric(38,0) not null	internal sequential id
uv_parent	numeric(38,0)	reference to sge_user.u_id
uv_time_start	timestamp without time zone	time interval for the validity of the reported value
uv_time_end	timestamp without time zone	time interval for the validity of the reported value
uv_variable	text	variable name
uv_svalue	text	variable value for string variables
uv_dvalue	double precision	variable value for numeric variables
uv_dconfig	double precision	configured capacity for consumables

Table 46: Information Available from `sge_user_values`

sge_job_class_values

The sge_job_class_values table holds name/values pairs related to job classes.

Attribute	Type	Description
jcv_id	numeric(38,0) not null	internal sequential id
jcv_parent	numeric(38,0)	reference to sge_job_class.jc_id
jcv_time_start	timestamp without time zone	time interval for the validity of the reported value
jcv_time_end	timestamp without time zone	time interval for the validity of the reported value
jcv_variable	text	variable name
jcv_svalue	text	variable value for string variables
jcv_dvalue	double precision	variable value for numeric variables
jcv_dconfig	double precision	configured capacity for consumables

Table 47: Information Available from `sge_job_class_values`

Sharetree Usage

sge_share_log

Attribute	Type	Description
sl_id	numeric(38,0) not null	
sl_curr_time	timestamp without time zone	
sl_usage_time	timestamp without time zone	
sl_node	text	
sl_user	text	
sl_project	text	
sl_shares	integer	
sl_job_count	integer	
sl_level	double precision	
sl_total	double precision	
sl_long_target_share	double precision	
sl_short_target_share	double precision	
sl_actual_share	double precision	
sl_usage	double precision	
sl_cpu	double precision	
sl_mem	double precision	
sl_io	double precision	
sl_ltcpu	double precision	
sl_ltmem	double precision	
sl_ltio	double precision	

Table 48: Information Available from `sge_share_log`

Index:

```
"sge\_share\_log\_pkey" PRIMARY KEY, btree (sl_id)
```

Statistics

sge_statistic

Attribute	Type	Description
s_id	numeric(38,0) not null	
s_name	text	

Table 49: Information Available from `sge_statistic`

Index:

```
"sge_statistic_pkey" PRIMARY KEY, btree (s_id)
```

Foreign Key Reference:

```
TABLE "sge_statistic_values" CONSTRAINT "sge_statistic_values_sv_parent_fkey"
   FOREIGN KEY (sv_parent) REFERENCES sge_statistic(s_id)
```

sge_statistic_values

Attribute	Type	Description
sv_id	numeric(38,0) not null	
sv_parent	numeric(38,0)	
sv_time_start	timestamp without time zone	
sv_time_end	timestamp without time zone	
sv_variable	text	
sv_dvalue	double precision	

Table 50: Information Available from `sge_statistic_values`

Index:

```
"sge_statistic_values_pkey" PRIMARY KEY, btree (sv_id)
"sge_statistic_values_idx0" btree (sv_parent, sv_variable, sv_time_end)
```

Foreign Key Constraints:

```
"sge_statistic_values_sv_parent_fkey" FOREIGN KEY (sv_parent)
   REFERENCES sge_statistic(s_id)
```

dbwriter Internal Data

sge_checkpoint

Attribute	Type	Description
ch_id	integer not null	
ch_line	integer	
ch_time	timestamp without time zone	

Table 51: Information Available from `sge_checkpoint`

Index:

```
"sge_checkpoint_pkey" PRIMARY KEY, btree (ch_id)
```

sge_version

Attribute	Type	Description
v_id	integer not null	
v_version	text not null	
v_time	timestamp without time zone	

Table 52: Information Available from `sge_version`

Index:

```
"sge_version_pkey" PRIMARY KEY, btree (v_id, v_version)
```

2 Common Tasks

2.1 Common Administrative Tasks in a Univa Grid Engine System

The following sections describe tasks commonly performed in a Univa Grid Engine system, including stopping a cluster, starting nodes, adding hosts, generating certificates, backing up and restoring a cluster.

2.1.1 Draining Then Stopping the Cluster

There are different reasons to drain a cluster or parts of cluster during the daily work with Univa Grid Engine. Old hosts that are removed from a cluster completely, service downtime of execution nodes or different software upgrades sometimes require that there are no running jobs

on corresponding hosts. Also major Univa Grid Engine updates might require that there are no pending jobs in the cluster or that certain types of jobs using specific features are not running.

The easiest way to get rid of those jobs would be to delete them, but the consequence of that approach is that the compute resources that were used by running jobs in the past would be lost. The alternative is to leave the jobs running until they end on their own. The following examples describe scenarios that could help in finding the best solution for draining a cluster or parts of a cluster.

Host Replacement

- No new jobs should be started on the host that is being replaced. To make the scheduler aware of this, disable the corresponding queues.

  # qmod -d “*@"

- Jobs that are already running on that host can continue. The state of those jobs can be observed with `qstat`.

  # qstat -s rs -l h=so02

- Once there are no more running jobs, the execution daemon can be shut down.

  # qconf -ke

- The host itself can now be shut down.

Minor Univa Grid Engine Upgrade Stipulating Certain Job Types Not Run During the Upgrade

- Create or enhance a server JSV script to detect and reject the job types that are not allowed in the system, and make this script active.

  # qconf -mconf ... jsv_url ...

- `qstat`, `qstat -j` in combination with some knowledge of the users' jobs will help find running instances of jobs that were submitted in the past.

- Once all running jobs finished, perform the upgrade.

2.1.2 Starting Up and Activating Nodes Selectively

One way to upgrade a Univa Grid Engine system is to use the backup/restore functionality to set up a second, identical cluster. This is described in the section Installation Guide -> Updating with Two Separate Clusters on the Same Resource Pool (Clone Configuration) in the Installation Guide. If this upgrade is done and the functionality of the old cluster is not disabled, two identical clusters will exist: the initial one can still be used, and the second one can be tested before it is made active. Compute resources can be disabled in the initial cluster selectively and enabled in the new cluster.

Disable all queues in the new cluster.

```
# qmod -d "*"
```

Deactivate old daemons, and activate compute resources in the new cluster.

- Disable a subset of resources in the old cluster.

```
# qmod -d "*@<hostname>"
# ...
```

- Wait till jobs that are still running have finished.

```
# qstat -s rs -l h=<hostname>
```

- Shutdown the execution daemon in the old cluster.

```
# qconf -ke <hostname>
```

- Enable the host in the new cluster.

```
# qconf -e "*@<hostname>"
```

- Make users aware that they can submit jobs into the new cluster. **Continue with the previous step as long as there are enabled compute resources in the old cluster. Now the old cluster can be uninstalled.**

2.1.3 Adding New Execution Hosts to an Existing Univa Grid Engine System

There are three cases in which a new execution host can be added to the Univa Grid Engine:

1. The current system is not Windows-enabled and not running in CSP mode, and the host to be added is not running Windows as operating system.

   In this case simply follow the instruction of section Installation Guide ->Execution Host installation - Execution Host Installation of the Installation Guide.

2. The current system is CSP enabled and the host to be added is not running Windows as operating system.

   In this case certificate files need to be transferred before the system can be installed. First execute the steps described in installation step *Transfer Certificate Files and Private Keys (Manually)* in section Installation Guide -> Interactive Installation - Interactive Installation of the Installation Guide, and then do the regular execution host installation described in Installation Guide -> Execution Host installation - Execution Host Installation.

3. The current system is not Windows-enabled and not running in CSP mode, and the goal is to add a Windows host.

   Then the system has to be Windows-enabled before it is possible to successfully install and use the additional Windows host. Find further instructions below.

   To Windows-enable a Univa Grid Engine system after the qmaster installation has already been done, execute the following steps:

- Enable the security framework.
  `# $SGE_ROOT/util/sgeCA/sge_ca -init`
- Make the new windows host an admin host.
  `# qconf -ah <new_windows_hosts>`
- Follow the steps of *Transfer Certificate Files and Private Keys (Manually)* in section Installation Guide -> Interactive Installation - Interactive Installation of the Installation Guide.
- Then do a regular execution host installation on the new Windows host as described in Installation Guide -> Execution Host installation - Execution Host Installation.

2.1.4 Generate/Renew Certificates and Private Keys for Users

The following steps represent an easy way to create the required certificates and private keys for those users that want to access a CSP secured Univa Grid Engine system:

Create a text with user entries.

- Create a text file containing one line for each user that should get access to the system.
- Each line has three entries separated by colon character (:) – UNIX username, full name, email address.

```
peter:Peter Yarrow:peter@univa.com
paul:Paul Stookey:paul@univa.com
mary:Mary Travers:mary@univa.com
```

Generate the files.

- Execute the following command, and specify the created text file as a parameter:

```
# $SGE_ROOT/util/sgeCA/sge_ca -usercert <text_file>
```

Check the results.

- Now the default location for user certificates will contain additional entries.

```
# ls -l /var/sgeCA/port${SGE_QMASTER_PORT}/${SGE_CELL}/userkeys
   ...
   dr-x------  2 peter staff        512 Mar 5 14:00 peter
   dr-x------  2 paul  staff        512 Mar 5 14:00 paul
   dr-x------  2 mary  staff        512 Mar 5 14:00 mary
   ...
```

Install the files.

- Security related files have to be installed in the `$HOME/.sge` directory of each user. Each user has to execute the following commands:

```
# . $SGE_ROOT/$SGE_CELL/common/settings.sh
# $SGE_ROOT/util/sgeCA/sge_ca -copy
Certificate and private key for user
<username> have been installed
```

Renew existing certificates:

Change the number of days that certificates should be valid.

- Modify the file `$SGE_ROOT/util/sgeCA/renew_all_certs.csh` to do so.

```
# extend the validity of the CA certificate by
set CADAYS = 365
# extend the validity of the daemon certificate by
set DAEMONDAYS = 365
# extend the validity of the user certificate by
set USERDAYS = 365
```

Renew the certificates.

```
# util/sgeCA/renew_all_certs.csh
```

Replace old certificates.

- The files in the directory `/var/sgeCA/...` need to be replaced. See the execution host installation description for more details.
- User certificates must also be replaced by each user (see section above).

The following examples provide common tasks to display or check certificates:

- To display a certificate:

```
# $SGE_ROOT/utilbin/${SGE_ARCH}/opensslx509 -in ~/.sge/port/${SGE_QMASTER_PORT}
   ${SGE_CELL}/certs/cert.pem -text
```

- To check the issuer:

```
# $SGE_ROOT/utilbin/${SGE_ARCH}/opensslx509 -issuer -in
   ~/.sge/port${SGE_QMASTER_PORT}${SGE_CELL}/certs/cert.pem -noout
```

- To show validity:

```
# $SGE_ROOT/utilbin/${SGE_ARCH}/opensslx509 -dates -in
   ~/.sge/port${SGE_QMASTER_PORT}${SGE_CELL}/certs/cert.pem -noout
```

- To show the fingerprint:

```
# $SGE_ROOT/utilbin/${SGE_ARCH}/opensslx509 -fingerprint -in
   ~/.sge/port${SGE_QMASTER_PORT}${SGE_CELL}/certs/cert.pem -noout
```

2.1.5 Backup and Restore the Configuration

During the backup process, all information concerning the configuration of a cluster is stored in a tar-file which can be used later for restoring. The backup saves configuration objects like queues, parallel environments, global/local cluster configuration. It also saves important files located under `$SGE_ROOT`, but it does *not* save information about pending or running jobs. Therefore the jobs will not be restored during the restoration process.

Creating a Manual Backup

To perform a backup manually, do the following steps.

Prepare the backup.

- Log in to an admin host as user root or as admin user.
- Source the settings file.

```
# source $SGE_ROOT/$SGE_CELL/common/settings.csh
```

- Start the backup process.

```
# cd $SGE_ROOT
# ./inst_sge -bup
```

Answer questions about the cluster.

- Enter the installation location.

```
SGE Configuration Backup
------------------------

This feature does a backup of all configuration you made
within your cluster.
Please enter your SGE_ROOT directory.
```

- Enter the cell name.

```
Please enter your SGE_CELL name.
```

- Enter the backup destination directory.

```
Where do you want to save the backup files?
```

- Should backup be compressed?

```
If you are using different tar versions (gnu tar/ solaris tar), this option
can make some trouble. In some cases the tar packages may be corrupt.
Using the same tar binary for packing and unpacking works without problems!

Shall the backup function create a compressed tar-package with your files? (y/n)
```

- Enter the backup file name.

```
Please enter a filename for your backup file.

configuration
sched_configuration
accounting
bootstrap
qtask
settings.sh
act_qmaster
sgemaster
settings.csh
...
local_conf/

... backup completed
All information is saved in
```

Verify. * Verify that the backup file was created.

Automating the Backup Process

The backup process can be automated. To do this, a backup template can be created. The `-auto` command line parameter causes the backup script to read all backup parameters from the template file instead of asking them interactively.

An example of a backup template can be found here:

`$SGE_ROOT/util/install_modules/backup_template.conf`:

```
################################################
# Autobackup Configuration File Template
################################################

# Please, enter your SGE_ROOT here (mandatory)
SGE_ROOT=""

# Please, enter your SGE_CELL here (mandatory)
SGE_CELL=""

# Please, enter your Backup Directory here
# After backup you will find your backup files here (mandatory)
# The autobackup will add a time /date combination to this dirname
# to prevent an overwriting!
BACKUP_DIR=""

# Please, enter true to get a tar/gz package
# and false to copy the files only (mandatory)
TAR="true"
```

```
# Please, enter the backup file name here. (mandatory)
BACKUP_FILE="backup.tar"
```

The automated backup process can be started with the following command:

```
inst_sge -bup -auto <backup_template>
```

There is no need to shut down the cluster during this operation.

Restoring from a Backup

The following steps are necessary to restore form a previous backup:

Prepare to restore.

- Log in to an admin host as user root or as admin user.
- Source the settings file.

```
# source $SGE_ROOT/$SGE_CELL/common/settings.csh
```

- Start the backup process.

```
# cd $SGE_ROOT
# ./inst_sge -rst
SGE Configuration Restore
-------------------------
This feature restores the configuration from a backup you made
previously.
```

Answer questions about the cluster. * Enter the installation directory.

```
Please enter your SGE_ROOT directory.
```

- Specify the cell name.

```
Please enter your SGE_CELL name.
```

- Was compression enabled during the backup process?

```
Is your backup file in tar.gz[Z] format?
```

- Specify the location of the backup file.

```
Please enter the full path and name of your backup file

configuration
sched_configuration
accounting
bootstrap
qtask
settings.sh
act_qmaster
sgemaster
settings.csh
sgeexecd
shadow_masters
cluster_name
jobseqnum
advance_reservations/
admin_hosts/
...
local_conf/
local_conf/su10.local
```

- Shut down `qmaster`, if it is running.

  ```
  Found a running qmaster on your masterhost: <qmaster_hostname>
  Please, check this and make sure, that the daemon is down during the restore!

  Shutdown qmaster and hit, <ENTER> to continue, or <CTRL-C> to stop
  the restore procedure!
  ```

- To shut down the master, open a new terminal window and trigger the shutdown before continuing with the restore.

  ```
  # qconf -km
  ```

Verify

- Verify the detected spooling method.

  Spooling Method: classic detected!

  Your configuration has been restored

- Restart `qmaster` as user root.
- Verify the Univa Grid Engine configuration.

2.1.6 Changing the Univa Grid Engine admin password for all UGE Starter Services on all execution hosts

Typically, company security policies demand to change the passwords of all users regularly, also the password of the Univa Grid Engine admin user. This password is registered locally on each

execution host for the “Univa Grid Engine Starter Service” that starts the execution daemon at host boot time. If the password of this user is changed on the Active Domain server, the locally registered password also must be changed for each “Univa Grid Engine Starter Service” in order to allow it to work properly. To avoid having to log on to each single execution host and change the password manually there, the following command allows to do this remotely from a single Windows host:

```
sc.exe \\exechost config "UGE_Starter_Service.exe" password= the_new_password
```

Where:

- “\\exechost” is the name of the Univa Grid Engine execution host, the name must be prepended by two backslashes.
- “the_new_password” is the new password of the Univa Grid Engine admin user - the blank between “password=” and the new password is important.

This command has to be started by a user that has sufficient permissions to execute the change - usually, this is a member of the “Domain Admins” group.

To change the password on all execution hosts, a file with the names of all execution hosts, each prepended by two backslashes and one name per line, must be prepared, e.g. file “exechosts.txt”:

```
\\hostA
\\hostB
\\hostC
```

Then this batch script can be used to apply the password change to all execution hosts:

```
@echo off
for /F %%i in (exechosts.txt) do (
   sc %%i config "UGE_Starter_Service.exe" password= the_new_password
)
```

2.2 Managing User Access

In a system where CSP mode is enabled, by default only users who have the necessary certificates and private keys have the right to submit and execute jobs in the Univa Grid Engine system.

Restrictions might be setup by an existing Univa Grid Engine administrator. Access restrictions can be setup in different Univa Grid Engine configuration objects to limit the access to the cluster, certain hosts/queues, certain job types or commands.

To increase the permissions for a user, it might be possible to make this user the owner of queues, make that user an operator or an administrator. After the installation of an Univa Grid Engine system, the only administrator in a cluster is the admin user. Details are explained in the sections below.

2.2.1 Setting Up a Univa Grid Engine User

To set up a user account for a Univa Grid Engine user that should be able to submit and execute jobs in a cluster, address the following requirements:

- The user needs a UNIX or Windows user account.
- The username has to be the same on all hosts that will be accessed by the user. For Windows hosts, this means the short username has to be the same like on the UNIX hosts and on each host the default AD domain has to be set properly, so the username without AD domain prefix resolves to the right user.
- It is also recommended that the user id and primary group id be the same on all UNIX hosts. The ids of UNIX and Windows hosts differ, of course.
- The user id has to be greater or equal to the *min_uid* and the primary group id has to be greater or equal to the *min_gid* so that the user has the ability to submit and execute jobs in a cluster. Both parameters are defined in the global configuration and have a value of 0 by default to allow root to run jobs, too. This does not apply for Windows hosts.
- In CSP enabled system, it is also necessary to take the steps described in section Generate/Renew Certificates and Private Keys for Users so that each user has access to a certificate and private keyfile to be able to use commands to submit jobs. In CSP enabled systems, it is not possible to use Windows execution, submit or admin hosts.
- Users need to be able to read the files in `$SGE_ROOT/$SGE_CELL/common`, and it is recommended to have full access to the directory referenced by `$TMPDIR`. On Windows hosts, instead of the $TMPDIR, the fixed directory "C:\tmp" is used.

To access certain Univa Grid Engine functionalities, additional steps are required:

- Advance Reservations

  Users are not allowed to create advance reservations by default. This feature has to be enabled by an administrator by adding the username to the *arusers* access list.

- Deadline Jobs

  To submit deadline jobs, users need to be able to specify the deadline initiation time. This is only allowed if an administrator adds the user name to the *deadlineusers* access list.

- Access Lists

  If access is additionally restricted in the cluster configuration, host configuration, queues, projects or parallel environments through the definition of access list, then users need to be added also to those access lists before access to corresponding parts of the cluster will be granted. For example, to be able to add a user to the share tree it is necessary to define that user in the Univa Grid Engine system. If projects are used for the share tree definition, that user should be given access to one project, otherwise the jobs of the user might be executed in the lowest possible priority class, which might not be intended.

2.2.2 Managers

Users that are managers have full access to an Univa Grid Engine cluster. All requirements that need to be fulfilled for regular users also apply to managers.

The admin user that is defined during the installation process of the Univa Grid Engine software is automatically a manager. This user class can execute administrative commands on administration hosts. Administrative commands are all the commands that change configuration parameters in a cluster or that change the state of configuration objects.

In contrast to other managers, the default admin user will also have file access to central configuration files.

Commands to Manage Managers

Managers can be added, modified or listed with the following commands:

- `qconf -sm`

Displays the list of all uses with administrative rights.

- `qconf -am <username>`

Makes the specified user a manager.

- `qconf -dm <username>`

Deletes the specified user from the list of managers.

2.2.3 Operators and Owners

Users that are defined as operators have the right to change the state of configuration objects but they are not allowed to change the configuration of a Univa Grid Engine cluster. For example, operators can enable or disable a queue, but not to change a configuration attribute like *slots* of a queue instance.

The same permissions and restrictions apply to queue owners, except the state changes they request will only be successful on those queues they own. Sometimes it makes sense to make users the owners of the queue instances that are located on the workstation they regularly work on. Then they can influence the additional workload that is executed on the machine, but they cannot administrate the queues or influence queues on different host. In combination with `qidle` command, it is for example possible to configure a workstation in a way so that the Univa Grid Engine workload will only be done when the owner of the machine is currently not working.

Commands to Manage Operators

Operators can be added, modified or listed with the following commands:

- `qconf -so`

Displays the list of all uses with operator rights.

- `qconf -ao <username>`

Makes the specified user an operator.

- `qconf -do <username>`

Deletes the specified user from the list of operators.

Commands to Manage Queue Owners

The owner list of a queue is specified by the queue field *owner_list*. The field specifies a comma separated list of login names of those users who are authorized. If it has the value *NONE*, then only operators and administrators can trigger state changes of the corresponding queue.

To modify or lookup the field value, use the following commands:

- `qconf -sq <queue_name>`

Prints all queue fields to stdout of the terminal where the command is executed.

- `qconf -mq <queue_name>`

Opens an editor so that queue fields can be modified.

2.2.4 Permissions of Managers, Operators, Job or Queue Owners

System Operations Or Configuration Changes

Operation	Job Owner	Queue Owner	Operator	Manager
Can add/change/delete UGE configuration settings of all configuration objects (e.g. hosts, queues, parallel environments, checkpointing objects . . .)	-	-	-	X
Can clear user and project usage in share tree. (see `qconf -clearusage`).	-	-	-	X
Can use the *forced* flag to trigger job state changes (see `qconf -f`)	-	-	-	X
Kill qmaster event clients (see `qconf -kec`)	-	-	-	X

Operation	Job Owner	Queue Owner	Operator	Manager
Kill/Start qmaster threads (see `qconf -kt | -at`)	-	-	-	X
Trigger scheduler monitoring (see `qconf -tsm`)	-	-	-	X
Add/delete users from ACLs (see `qconf -au | Au | -du`)	-	-	X	X

Table 53: Permissions of Managers, Operators, Job or Queue Owners

Queue Operations

Operation	Job Owner	Queue Owner	Operator	Manager
Clear error state of queue instance (see `qconf -c`)	-	X^1	X	X
Enable/Disable any queue instance (see `qconf -d | -e`)	-	X^1	X	X
Suspend/unsuspend any queue instance (see `qconf -s | -us`).	-	X^1	X	X

Table 54: Permissions of Managers, Operators, Job or Queue Owners

Job Operations

Operation	Job Owner	Queue Owner	Operator	Manager
Change the AR request of jobs submitted by other users (see `qalter -ac`)	-	-	X	X
Change override tickets of other jobs (see `qalter -o`).	-	-	X	X
Change functional shares of other jobs (see `qalter -js`)	- .	-	X	X
Change attributes of jobs that were submitted by other users.	-	-	X	X
Increase priority of jobs owned by other users (see `qalter -p`).	-	-	X	X
Reschedule jobs of other users (see `qmod -rj | -rq`).	-	X^2	X	X
Suspend/unsuspend any job	-	X^2	X	X

Operation	Job Owner	Queue Owner	Operator	Manager
Clear error state of other jobs (see `qmod -c`).	-	-	X	X
Delete Job of other users (see `qdel`).	-	-	X	X
Set/unset system hold state of a job (see `qalter -h s|S`).	-	-	-	X
Set/unset operator hold state of a job (see `qalter -h o|O`).	-	-	X	X
Set/unset user hold state of a job (see `qalter -h u|U`).	X	-	X	X

Table 55: Permissions of Managers, Operators, Job or Queue Owners

1 These operations can only be applied on queues where the user is owner of.

2 Queue owners can not trigger state changes of jobs directly but the state changes are applied to running jobs when the owner of a queue triggers a state change of the owned queue where a job is running.

2.2.5 User Access Lists and Departments

User access lists are lists of user names that can be attached to configuration parameters of the following objects.

- Cluster Configuration
- Host
- Queue
- Projects
- Parallel Environment

The configuration parameters with the name *user_lists* and *acl* define access lists of which users will get access to the corresponding object, whereas the attributes with the name *xuser_lists* and *xacl* will define the access lists for those users who will *not* get access. A user that is referenced in both *user_lists* and *xuser_lists* or in both *acl* and *xacl* will not get access, whereas when both lists are set to *NONE* anyone can access the corresponding object.

The term *access* has different meanings for the different objects. Denied access in the cluster configuration means the user cannot use the whole cluster, whereas denied access to parallel environments will cause the Univa Grid Engine scheduler to skip scheduling of corresponding jobs when a user explicitly requests that parallel environment.

Note that *access lists* are also used as *department* in Univa Grid Engine. In contrast to access lists, users can be part of only of one department. Departments are used in combination with the function and override policy scheme.

The *type* field of an access list object defines if the corresponding object can be used as department or only as access list.

Commands to Add, Modify Delete Access Lists

Access lists can be added, modified or listed with the following commands:

- `qconf -sul`

Displays the names of all existing access lists.

- `qconf -dul <listname>`

Deletes the access list with the given name.

- `qconf -au <user> <user> ... <listname>`

Adds the specified users to the access list.

- `qconf -du <user> <user> ... <listname>`

Deletes the specified user from the access list.

- `qconf -am <listname>`

Opens an editor to modify the access list parameters.

- `qconf -Au <filename>`

Similar to `-au` with the difference that configuration is taken from file.

- `qconf -Mu<filename>`

Similar to `-mu` with the difference that configuration is taken from file.

Configuration Parameters of Access Lists

Each access list object supports the following set of configuration attributes:

Attribute	Value Specification
`name`	The name of the access list.
`type`	The type of the access list, currently one of ACL, or DEPT, or a combination of both in a comma separated list. Depending on this parameter, the access list can be used as access list only or as a department.

Attribute	Value Specification
`oticket`	The number of override tickets currently assigned to the department.
`fshare`	The current functional shares of the department.
`entries`	The entries parameter contains the comma separated list of user names or primary group names assigned to the access list or the department. Only a user's primary group is used; secondary groups are ignored. Only symbolic names are allowed. A group is differentiated from a user name by prefixing the group name with a '@' sign. When using departments, each user or group enlisted may only be enlisted in one department, in order to ensure a unique assignment of jobs to departments. For jobs without users who match any of the users or groups enlisted under entries, the `defaultdepartment` is assigned, if existing.

Table 56: Access List Configuration Attributes

2.2.6 Projects

Project objects are used in combination with the Univa Grid Engine policy scheme to express the importance of a group of jobs submitted as part of that project compared to other jobs in other projects. Details for the setup of the policy scheme can be found in section Managing Priorities and Usage Entitlements of the Administration Guide. The following sections describe the available commands and object attributes.

Commands to Add, Modify or Delete Projects

Access lists can be added, modified or listed with the following commands:

- `qconf -aprj`

Adds a new project.

- `qconf -Aprj <filename|dirname>`

Adds a new project that is defined in the specified file. If a directory is given, projects for every configuration file in that directory are added.

- `qconf -dprj <project_name>`

Deletes an existing project.

- `qconf -mprj <project_name>`

Opens an editor so that the specified project can be modified.

- `qconf -Mprj <filename|dirname>`

Modifies the project. New object configuration is taken from the specified file. If a directory is given, projects for every configuration file in that directory are modified.

- `qconf -sprj`

Shows the current configuration of the project.

- `qconf -sprjl`

Shows all existing projects of an Univa Grid Engine cluster.

- `qconf -sprjld [<project_list>]`

Shows a detailed list of all projects of an Univa Grid Engine cluster or projects in a given project list.

Configuration Parameters of Projects

Each project object supports the following set of configuration attributes:

Attribute	Value Specification
`name`	The name of the project.
`oticket`	The number of override tickets currently assigned to the project.
`fshare` `acl`	The current functional share of the project. A list of user access lists referring to those users being allowed to submit jobs to the project. If set to *NONE* all users are allowed to submit to the project except for those listed in *xacl*.
`xacl`	A list of user access lists referring to those users that are not allowed to submit jobs to the project.

Table 57: Project Configuration Attributes

2.3 Understanding and Modifying the Cluster Configuration

The host configuration attributes control the way a Univa Grid Engine cluster operates. These attributes are set either globally through the global host configuration object, or in a local host configuration object that overrides global settings for specific hosts.

2.3.1 Commands to Add, Modify, Delete or List Global and Local Configurations

Global and local configurations can be added, modified, deleted or listed with the following commands:

- `qconf -Aconf <filename>`

Adds a new local configuration that is defined in the specified file.

- `qconf -Mconf <filename>`

Modifies a local configuration that is defined in the specified file.

- `qconf -aconf <host>`

Adds a new local configuration for the given host.

- `qconf -dconf <host>`

Deletes an existing local configuration.

- `qconf -mconf <host> | global`

Modifies an existing local or global configuration.

- `qconf -sconf <host> | global`

Displays the global or local configuration.

- `qconf -sconfl <host> | global`

Displays the list of existing local configurations.

2.3.2 Configuration Parameters of the Global and Local Configurations

The global object and each local configuration object support the following set of configuration attributes. Note that the list is not complete. Find the full description in the man page `sge_conf(1)`.

Attribute	Value Specification
`execd_spool_dir`	The execution daemon spool directory. For Windows execution hosts, this always has to be the value "/execd_spool_dir/win-x86/placeholder" which is replaced on the Windows execution host by the corresponding path that is defined in the $SGE_ROOT/$SGE_CELL/common/path_map file.
`mailer`	Absolute path to the mail delivery agent.
`load_sensor`	A comma separated list of executables to be started by execution hosts to retrieve site configurable load information.

Attribute	Value Specification
`prolog` `epilog`	Absolute path to executables that will be executed before/after a Univa Grid Engine job.
`shell_start_mode`	Defines the mechanisms which are used to invoke the job scripts on execution hosts.
`min_uid` `min_gid`	Defines the lower bound on user/group IDs that may use the cluster.
`user_lists` `xuser_lists`	User access lists that define who is allowed access to the cluster.
`administrator_mail`	List of mail addresses that will be used to send problem reports.
`project` `xproject`	Defines which projects are granted access and where access will be denied.
`load_report_time`	The system load of execution hosts is periodically reported to the master host. This parameter defines the time interval between load reports.
`reschedule_unknown`	Determines whether jobs on execution hosts in an unknown state are rescheduled and thus sent to other hosts.
`max_unheard`	If the master host could not be contacted or was not contacted by the execution daemon of a host for `max_unheard` seconds, all queues residing on that particular host are set to status unknown.
`loglevel`	Defines the detail level for log messages.
`max_aj_instances`	This parameter defines the maximum number of array tasks to be scheduled to run simultaneously per array job.
`max_aj_tasks`	Defines the maximum number of tasks allowed for array jobs. If this limit is exceeded, then the job will be rejected during submission.
`max_u_jobs`	The number of active jobs each user can have in the system simultaneously.
`max_jobs`	The number of active jobs in the system.
`max_advance_reservations`	The maximum number of active advance reservations allowed in Univa Grid Engine.
`enforce_project`	When set to true, users are required to request a project during submission of a job.
`enforce_user`	When set to `true`, users must exist within the Univa Grid Engine system before they can submit jobs. `auto` means that the user will be automatically created during the submission of the job.

`auto_user_delete_time`
`auto_user_default_project`
`auto_user_fshare`
`auto_user_oticket`
	Defines different aspects for automatically created users.

Attribute	Value Specification
gid_range	Comma separated list of range expressions specifying additional group IDs that might be used by execution daemons to tag jobs.

Table 58: Project Configuration Attributes

2.4 Understanding and Modifying the Univa Grid Engine Scheduler Configuration

The Univa Grid Engine scheduler determines which jobs are dispatched to which resources. It runs periodically in a pre-defined interval, but can also be configured so that additional scheduling runs are triggered by job submission and job finishing events.

Crucial steps within a scheduler run are as follows:

- Create the job order list out of the pending job list.
- Create a queue instance order list based on a host sort formula or a sequence numbering schema (or both).
- Dispatch the jobs (based on the job-order list) to the resources (based on the queue-instance order list).

The scheduler configuration is a crucial part of each installation due to its influence on the overall cluster-utilization, job-throughput, and master host load. Univa Grid Engine offers a large set of variables, making the configuration very flexible.

Because this scheduler configuration section intersects with several other topics (such as the policy configuration), it is recommended to read all of the following sections and man pages: `man sched_conf, man sge_priority` and 'Special Activities -> Managing Priorities and Usage Entitlements - Managing Priorities and Usage Entitlements'.

2.4.1 The Default Scheduling Scheme

The scheduler configuration is printed with the `qconf -ssconf` command. Modify the scheduler configuration with the editor configured in the $EDITOR environment variable with the `qconf -msconf` command. The default configuration after a installation is shown below:

```
> qconf -ssconf
algorithm                         default
schedule_interval                 0:0:10
maxujobs                          0
job_load_adjustments              np_load_avg=0.15
load_adjustment_decay_time        0:7:30
host_sort_formula                 np_load_avg
```

```
schedd_job_info                 false
flush_submit_sec                0
flush_finish_sec                0
params                          none
reprioritize_interval           00:00:40
halftime                        168
usage_weight_list               wallclock=0.000000,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                 0
default_duration                INFINITY
weight_host_affinity            0.000000
weight_host_sort                1.000000
weight_queue_affinity           0.000000
weight_queue_host_sort          1.000000
weight_queue_seqno              0.000000
```

The scheduler parameters are explained in the table below:

Attribute	Value Specification
`algorithm`	The algorithm can't be changed; it is `default`.
`schedule_interval`	Specifies at which time interval the scheduler is called. The format is hours:minutes:seconds.
`maxujobs`	The maximum number of user jobs running at the same time. Note: 0 indicates that there is no limit. Since the advent of resource quota sets, configuring the user limit there is preferred because of its superior flexibility.

Attribute	Value Specification
`job_load_adjustment`	Determines the load correction (additional artificial load for the scheduler), that each job contributes to the machine load values after the job was dispatched. This avoids overloading a currently unloaded host by dispatching too many jobs on it, because load reporting is sluggish (right after scheduling, there is no additional load).
`load_adjustment_decay_time`	The load adjustment is scaled linearly. This means right after dispatching the job, the `job_load_adjustment` adjusts the load value of the resources with 100% influence. After a while the influence is reduced linearly until `load_adjustment_decay_time` is reached.
`host_sort_formula` l e b	An algebraic expression used to derive a single weighted oad value from all or part of the load parameters for ach host and from all or part of the consumable resources eing maintained for each host. The default is `np_load_avg` the normalized average load. t is only considered if `weight_host_sort` is not 0.
`sched_jobinfo`	If set to `true`, additional scheduling information can be seen in the `qstat -j` output. The default value is `false`, because is impacts the overall scheduler performance in bigger clusters.
`flush_submit_sec`	If unequal to 0, it defines an additional scheduler run that is performed the specified number of seconds after a job was submitted.
`flush_finish_sec`	If unequal to 0, it defines an additional scheduler run that is performed the specified number of seconds after a job finishes.
`params`	Additional parameters for the Univa Grid Engine scheduler: * DURATION_OFFSET : assumed offset between run-time of a job and the run-time from scheduler perspective * PROFILE : 1 = turning run-time profiling on * MONITOR : 1 = turning additional monitoring on * PE_RANGE_ALG : alternative behavior when selecting slots depending on a PE range
`other parameters`	see Managing Priorities and Usage Entitlements

Table 59: Scheduler Configuration Attributes

2.5 Configuring Properties of Hosts and Queues

Both hosts and queues offer a wide range of resources for jobs. While hosts are a common physical concept, queues can be seen as job containers spanning across multiple hosts. A specific queue on a specific host is called queue instance, which is a central element in Univa Grid Engine. One host can be part of multiple queues. Resources can be defined on a host level or on a queue level. This section describes the configuration of hosts and queues, as well as their intersection with complexes and load sensors.

2.5.1 Configuring Hosts

Univa Grid Engine hosts have two configurations: the **local cluster configuration** (also called execution host local configuration) and the **execution host configuration**.

Local Cluster Configuration

The local cluster configuration can override values from the global cluster configuration (`qconf -sconf`), to adapt them to the execution hosts' characteristics (like the path to the mailer or `xterm` binary). The following table lists the commands used to alter the local cluster configuration:

Attribute	Value Specification
`qconf -sconfl`	Shows all hosts with a local cluster configuration.
`qconf -sconf <hostlist>`	Shows the local cluster configuration of hosts from the .
'qconf -mconf '	Opens an editor and let the user configure the local cluster configurations of hosts in the .
`qconf -Mconf <hostlist>`	Modifies the local configuration
`qconf -aconf <hostlist>`	Adds new local cluster configurations to hosts given by a host list.
`qconf -Aconf <filelist>`	Adds new local cluster configurations to hosts given by a file list
`qconf -dconf`	Deletes the local cluster configuration of the host given by the host list.

Table 60: Local Cluster Configuration

The following attributes can be used for overriding the global cluster configuration:

- `execd_spool_dir`
- `mailer`
- `xterm`
- `load_sensor`
- `prolog`
- `epilog`
- `load_report_time`
- `rescheduler_unknown`
- `shepherd_cmd`
- `gid_range`
- `execd_params`

- qlogin_daemon
- qlogin_command
- rlogin_daemon
- rlogin_command
- rsh_daemon
- rsh_command
- libjvm_path
- additional_jvm_args

More details about these attributes can be found in the man page sge_conf and in the section Understanding and Modifying the Cluster Configuration.

Execution Host Configuration

The **execution host configuration** is modified with qconf -me <hostname>. Scripts should call qconf -Me <filename|dirname>, which allows changes to the configuration based on a given file or directory. The configuration can be shown with qconf -se <hostname> or qconf -seld. The following table illustrates the configuration host attributes.

Attribute	Value Specification
qconf -ae [<template>]	Adds a new execution host configuration, optionally based on a configuration template.
qconf -Ae <filelist|dirname>	Adds an execution host configuration based on a file. If a directory is specified, execution hosts for every configuration file in that directory are added.
qconf -de <hostlist>	Deletes execution host configuration based on the given host list.
qconf -me <hostname>	Modifies the execution host configuration of the host given by the hostname.
qconf -Me <filename|dirname>	Modifies an execution host given based on a configuration file. If a directory is specified, execution hosts for every configuration file in that directory are modified.
qconf -se <hostname>	Shows the execution host configuration of the given host.
qconf -sel	Shows all existing execution hosts of an Univa Grid Engine cluster.
qconf -seld [<hostlist>]	Shows a detailed list of all execution hosts of an Univa Grid Engine cluster or hosts in a given host list.

Table 61: TABLE: Execution Host Configuration

The following is an example of an execution host configuration:

```
> qconf -se ma
> csuse
hostname              macsuse
load_scaling          NONE
complex_values        NONE
load_values           arch=lx-amd64,num_proc=1,mem_total=1960.277344M, \
                      swap_total=2053.996094M,virtual_total=4014.273438M, \
                      load_avg=0.280000,load_short=0.560000, \
                      load_medium=0.280000,load_long=0.320000, \
                      mem_free=1440.257812M,swap_free=2053.996094M, \
                      virtual_free=3494.253906M,mem_used=520.019531M, \
                      swap_used=0.000000M,virtual_used=520.019531M, \
                      cpu=2.900000,m_topology=SC,m_topology_inuse=SC, \
                      m_socket=1,m_core=1,m_thread=1,np_load_avg=0.280000, \
                      np_load_short=0.560000,np_load_medium=0.280000, \
                      np_load_long=0.320000
processors            1
user_lists            NONE
xuser_lists           NONE
projects              NONE
xprojects             NONE
usage_scaling         NONE
report_variables      NONE
```

Execution Host Configuration Fields:

- The `hostname` field denotes the name of the host.
- With `load_scaling`, load values can be transformed. This can be useful when standardizing load values based on specific host properties (e.g. number of CPU cores). More information and examples about load scaling are in the Special Activities Guide in section Scaling the Reported Load.
- The `complex_values` field is used to configure host complexes. More details about this field are described in the [Utilizing Complexes and Load Sensors][Utilizing Complexes and Load Sensors].
- The `load_values` and the `processors` field are read-only, and they can only be seen with `qconf -se <hostname>`. these fields are not available when the execution host configuration is modified.
- The `usage_scaling` provides the same functionality as `load_scaling`, but with the difference that it can only be applied to the usage values **mem**, **cpu**, and **io**. When no scaling is given, the default scaling factor (`1`) is applied.
- Access control can be configured on user and project level.
  - `user_lists` and `xuser_lists` contain a comma-separated list of access lists (see also `man access_lists`. The default value of both fields is `NONE`, which allows any user access to this host. If access lists are configured in `user_lists`, only users within this list (but not listed in `xuser_lists`), have access to the host.

  - All users in the access lists of `xuser_list` have no access.
  - Inclusion and exclusion of jobs based on the projects they are associated with is configured in the `projects` and `xprojects` field. They contain a comma-separated list of projects that are allowed or disallowed on the specific host. By default (both values `NONE`), all projects are allowed. If a project is listed in both lists, access is disallowed for all jobs of this project.

- The `report_variables` field contains a list of load values that are written in the reporting file each time a load report is sent from the execution daemon to the qmaster process.

Administrative and Submit Hosts

Univa Grid Engine allows the administrator to control from which hosts it is allowed to submit jobs, and which hosts can be used for administrative tasks, such as changing configurations.

The following table shows all commands used for configuring the administrative host list.

Attribute	Value Specification
`qconf -ah <hostnamelist>`	Adds one or more host to the administrative host list.
`qconf -dh <hostnamelist>`	Deletes one or more hosts from the list of administrative hosts.
`qconf -sh`	Show all administrative hosts.

Table 62: Admin Host Configuration

Submit hosts are configured in a similar way. The following table shows the commands used for configuring the submission host list.

Attribute	Value Specification
`qconf -as <hostnamelist>`	Adds one or more host to the submit host list.
`qconf -ds <hostnamelist>`	Deletes one or more hosts from the list of submit hosts.
`qconf -ss`	Show all submits hosts.

Table 63: Submission Host Configuration

On Windows, additionally the dynamic link library $SGE_ROOT/lib/win-x86/pthreadVC2.dll has to be copied to the Windows directory (usually "C:\Windows") on each submit or admin host, in order to make the Univa Grid Engine binaries work on these hosts.

Grouping of Hosts

To simplify the overall cluster configuration, Univa Grid Engine hosts can be arranged with the

host-group feature. Host-groups allow the administrator and the user to identify a group of hosts just with a single name. To differentiate host names from host-group names, host-group names always start with the @ prefix.

Attribute	Value Specification
`qconf -ahgrp <group>`	Adds a new host group entry with the name and opens an editor for editing.
`qconf -Ahgrp <filename>`	Adds a new host group entry with the configuration based on the file .
`qconf -dhgrp <group>`	Deletes the host group with the name .
`qconf -mhgrp <group>`	Modifies the host group in an interactive editor session.
`qconf -Mhgrp <filename>`	Modifies a host group based on a configuration file .
`qconf -shgrp <group>`	Shows the configuration of the host-group .
`qconf -shgrp_tree <group>`	Shows the host-group with sub-groups in a tree structure.
`qconf -shgrp_resolved <group>`	Shows the host-group with an resolved host-list.
`qconf -shgrpl`	Shows a list of all host-groups.

Table 64: Host Group Configuration

An host-group configuration consists of two entries:

- The `group_name`, that must be a unique name with an "@" prefix
- A `hostlist`, that can contain host-names and/or other host-group names. Having host-group names in the hostlist allows one to structure the hosts within a tree. The following example points this out.

Example: Grouping Host-Groups in a Tree Structure

In the first step, the lowest host-groups with real host-names must be added:

```
> qconf -ahgrp @lowgrp1
group_name @lowgrp1
hostlist host1

> qconf -ahgrp @lowgrp2
group_name @lowgrp2
hostlist host2

> qconf -ahgrp @lowgrp3
group_name @lowgrp3
hostlist host3
```

```
> qconf -ahgrp @lowgrp4
group_name @lowgrp4
hostlist host4
```

Now the mid-level groups can be defined:

```
> qconf -ahgrp @midgrp1
group_name @midgrp1
hostlist @lowgrp1 @lowgrp2

> qconf -ahgrp @midgrp2
group_name @midgrp2
hostlist @lowgrp3 @lowgrp4
```

In a final step, the highest host-group is added:

```
> qconf -ahgrp @highgrp
group_name @highgrp
hostlist @midgrp1 @midgrp2
```

Show the tree:

```
> qconf -shgrp_tree @highgrp
@highgrp
   @midgrp1
      @lowgrp1
         host1
      @lowgrp2
         host2
   @midgrp2
      @lowgrp3
         host3
      @lowgrp4
         host4
```

The resolved host-list looks like the following:

```
> qconf -shgrp_resolved @highgrp
host1 host2 host3 host4
```

2.5.2 Configuring Queues

Queues are job-containers that are used for grouping jobs with similar characteristics. Additionally, with queues, priority-groups can be defined with the subordination mechanism. A queue must have a unique queue name that is set with the `qname` attribute, and span over a defined set of hosts (`hostlist`). The `hostlist` can contain `none` for no host, `@all` for all hosts, or a list of hostnames and/or host group names. The following table gives an overview over the queue configuration commands.

Attribute	Value Specification
`qconf -aq [qname]`	Adds a new queue.
`qconf -Aq <filename>`	Adds a new queue based on the configuration given by the file `filename`.
`qconf -cq <queuelist>`	Cleans a queue from jobs. The queues are given in the .
`qconf -dq <queuelist>`	Deletes one or more queues. The name of queues are given in the .
`qconf -mq <hostname>`	Displays an editor for modifying a queue configuration.
`qconf -Mq <filename>`	Modifies a queue configuration based on a configuration file.
`qconf -sq <queuelist>`	Shows the queue configuration for one or more queues. If no parameter is given, a queue template is shown.
`qconf -sql`	Shows a list of all configured queues.

Table 65: Queue Configuration Commands

Example: Adding a New Queue, Showing the Queue Configuration and Deleting the Queue

```
> qconf -aq new.q
 qname                 new.q
 hostlist              @allhosts
 ...
 (closing the vi editor with CTRL-ZZ)
 user@host added "new.q" to cluster queue list

> qconf -sq new.q
qname                 new.q
hostlist              @allhosts
seq_no                0
load_thresholds       np_load_avg=1.75
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
rerun                 FALSE
slots                 1
tmpdir                /tmp
shell                 /bin/sh
prolog                NONE
```

```
epilog                NONE
shell_start_mode      posix_compliant
starter_method        NONE
suspend_method        NONE
resume_method         NONE
terminate_method      NONE
notify                00:00:60
owner_list            NONE
user_lists            NONE
xuser_lists           NONE
subordinate_list      NONE
complex_values        NONE
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                INFINITY
h_core                INFINITY
s_rss                 INFINITY
h_rss                 INFINITY
s_vmem                INFINITY
h_vmem                INFINITY

> qconf -dq new.q
user@host removed "new.q" from cluster queue list
```

Queue Configuration Attributes

The queue configuration involves a spectrum of very different settings. For more detailed information, see `man queue_conf`.

Queue Limits

The queue configuration allows one to define a wide range of limits. These limits (by default, `INFINITY`) limit the following parameters of a job running in this particular queue:

- runtime (`h_rt/s_rt`)
- CPU time (`h_cpu/s_cpu`)
- number of written disc blocks (`h_fsize/s_fsize`)

- data segment size (`h_data/s_data`)
- stack size (`h_stack/s_stack`)
- maximum core dump file size (`h_core/s_core`)
- resident set size (`h_rss/s_rss`)
- virtual memory size (`h_vmem/s_vmem`)

All limits are available as soft and hard limit instances (prefix `s_` and `h_`).

The following table shows the meaning of the different limits:

Attribute	Value Specification
`h_rt`	Limits the real time (wall clock time) the job is running. If a job runs longer than specified a SIGKILL signal is sent to the job.
`s_rt`	The soft real time limit (wall clock time limit) warns a job with a catchable SIGUSER1 signal, if exceeded. After a defined time period (see `notify` parameter), the job is killed.
`h_cpu`	Limits the CPU time of a job. If a job needs more CPU time than specified, then the job is signaled with a SIGKILL. In case of parallel jobs, this time is multiplied by the number of granted slots.
`s_cpu`	Limits the CPU time of a job. If a job needs more CPU time than specified, then the job is signaled with SIGXCPU, which can be caught by the job. In case of parallel jobs, this time is multiplied by the number of granted slots.
`h_vmem`	The virtual memory limit limits the total amount of combined memory usage of all job processes. If the limit is exceeded a SIGKILL is sent to the job.
`s_vmem`	The virtual memory limit limits the total amount of combined memory usage of all job processes. If the limit is exceeded a SIGXCPU is sent, which can be caught by the job.
`h_fsize, s_fsize, h_data, s_data, h_stack, s_stack, h_core, s_core, h_rss`,s_rss‘	These limits have the semantic of the `setrlimit` system call of the underlying operating system.

Table 66: Queue Resource Limits

Queue Sequencing and Thresholds

The `seq_no` field denotes the sequence number the queue (or the queue instances) has when the queue sort method of the scheduler configuration is based on sequence numbers. More information can be found in the scheduler configuration section.

With `load_thresholds` is possible to define when an overloaded queue instance is set to the `alarm` state. This state prevents more jobs from being scheduled in this overloaded queue instance.

`suspend_thresholds` defines the thresholds until the queue is set into a suspended state. The default value is `NONE` (no threshold). If suspend thresholds are defined, and one of the thresholds is exceeded, within the next scheduling run, a pre-defined number of jobs running in the queue are suspended. The number of suspended jobs is defined in the `nsuspend` field. The `suspend_interval` field denotes the time interval until the next `nsuspend` number of jobs are suspended, in case one of the suspend thresholds remains exceeded.

Queue Checkpoints, Processing and Type

The `priority` value specifies at which operating system process priority value the jobs are started. The possible range for priority (also called **nice** values) is from -20 to 20, where -20 is the highest priority and 20 the lowest priority. This value only has effect when dynamic priority values are turned off (i.e. `reprioritize` is false in the global cluster configuration).

The `min_cpu_interval` defines the time interval between two automatic checkpoints. Further information about checkpointing can be found in the man page `sge_ckpt`.

The `processors` field can be used to use a pre-defined processor set on the Solaris operating system. It is deprecated since the advent of the core binding feature.

Warning

Do not use the processors field when using the core binding feature on Solaris!

The `qtype` field specifies what type the queue has. Allowed values are `BATCH`, `INTERACTIVE`, a combination of both and `NONE`. Interactive queues can run jobs from interactive commands, like `qrsh`, `qsh`, `qlogin`, and `qsub -now y`. The remaining batch jobs can only run in `BATCH` queues.

The list of checkpointing environments associated with this queue can be set at `ckpt_list`. Further information about checkpointing can be found in the man page `sge_ckpt`.

The `pe_list` contains a list of parallel environments which are associated with this queue.

If `rerun` is set to `FALSE` (the default value), the behavior of the jobs running in the queue is that they are not restarted in case of an execution host failure (see `man queue_conf` for more details). If set to `TRUE`, the jobs are restarted (run again) in such a case. The specified default behavior for the jobs can be overruled on job level, when the `-r` option is used during job submission.

The `slots` field defines the number of job slots that can be used within each queue instance (a queue element on a host). In case only normal (sequential) jobs are running within the queue, it denotes the number of jobs each queue instance is capable to run. When the queue spans over `n` hosts, the whole queue is limited to `n*slots-value` slots.

Queue Scripting

The `tmpdir` field specifies the path to the host's temporary file directory. The default value is `/tmp`. When the execution daemon starts up a new job, a temporary job directory is created in `tmpdir` for this particular job, and the job environment variables `TMP` and `TMPDIR` are set to this path.

The `shell` field points to the command interpreter, which is used for executing the job script. This shell is only taken into account when the `shell_start_mode` in the cluster configuration is

set to either `posix_compliant` or `script_from_stdin`. This parameter can also be overruled by the `-S` parameter on job submission time.

The `prolog` field can be set to a path to a shell script that is executed before a job running in this queue starts. The shell script is running with the same environment as the job. The output (stdout and stderr) is redirected to the same output file as the job. Optionally the user under which the prolog script is executed can be set with a `<username>@` prefix.
For Docker jobs, the `prolog` is first executed on the physical host, then in the container. In the container, the environment variable $SGE_IN_CONTAINER is set (always to the value "1") to allow the script to distinguish where it was started. In the container, the `prolog` always runs under the job user, the `<username>@` prefix is ignored there.

The `epilog` field sets the path to a shell script that is executed after a job running in this queue ends. Also see the `prolog` field and the `queue_conf` man page.
For Docker jobs,the `epilog` is first executed in the container, then on the physical host. Here, $SGE_IN_CONTAINER is set and the `<username>@` prefix is ignored in the container, too.

The `shell_start_mode` determines which shell executes the job script. Possible values are `posix_compliant` (take the shell specified in `shell` or on job submission time), `unix_behavior` (take the shell specified within the shell script (#!) or on job submission time), or `script_from_stdin`. More detailed information can be found in the `queue_conf` man page.

The `starter_method` allows one to change the job starting facility. Instead of using the specified shell, the configured executable is taken for starting the job. By default, this functionality is disabled (value `NONE`).

Queue Signals and Notifications

Univa Grid Engine suspends, resumes, and terminates the job process usually by default with the signals `SIGSTOP`, `SIGCONT`, and `SIGKILL`. These signals can be overridden with the queue configuration parameters `suspend_method`,`resume_method`, and `terminate_method`. Possible values are signal names (like `SIGUSR1`) or a path to an executable. The executable can have additional parameters. Special parameters are `$host`, `$job_owner`, `$job_id`, `$job_name`, `$queue`, and `$job_pid`. These variables are substituted with the corresponding job specific values.

When a job is submitted with a `-notify` option, the `notify` field in the queue configuration defines the time interval between the delivery of the notify signal (`SIGUSR1, SIGUSR2`) and the suspend/kill signal.

Queue Access Controls and Subordination

If user names are listed in the `owner_list` queue configuration attribute, these users have the additional right to disable or suspend the queue.

Access control to the queue is configured by the `user_lists`, `xuser_lists`, `projects`, and `xprojects` lists. More detailed information about configuration of these fields can be found in the man page `access_lists` and in the section 'Configuring Hosts - Configuring Hosts'.

Queue-wise and slot-wise subordination can be defined in the `subordinate_list`. More information about the subordination mechanism can be found in the Special Activities Guide in section 'Special Activities -> Implementing Pre-emption Logic - Implementing Pre-emption Logic'.

Queue Complexes

If a previously declared complex (see man complex) should be used as a queue complex or queue consumable (i.e. available on queue instance level), it must be initialized in the `complex_values` field.

Queue consumables and queue complexes must be initialized on the queue level. The `complexes_values` field allows to configure the specific values of the complexes (e.g. `complex_values test=2` sets the complex test to the value 2 on each queue instance defined by the queue).

Queue Calendar and State

The `calendar` attribute associates a queue with a specific calendar that controls the queue. More information about calendars can be found in the man page `calendar_conf`.

The `initial_state` field specifies the state of the queue instances have after an execution daemon (having this queue configured) is starting or when the queue is added the first time. Possible values are `default`, `enabled`, and `disabled`.

2.5.3 Utilizing *Complexes* and *Load Sensors*

The **complexes** concept in Univa Grid Engine is mainly used for managing resources. The load sensors are used on execution hosts to provide a functionality for reporting the state of resources in a flexible way. The following sections describe both concepts and show examples of how they can be used for adapting Univa Grid Engine to the needs of users.

Configuring Complexes

Complexes are an abstract concept for configuring and denoting resources. They are declared in the complex configuration (`qconf -mc`). Depending on if these complexes are initialized with a value, they can reflect either **host resources** or **queue resources**. Host based complexes are initialized in the `complex_values` field of this execution host configuration (`qconf -me <hostname>`). If they are initialized in the global host configuration (`qconf -me global`), they are available in the complete cluster. The configuration value is a list of name/value pairs that are separated by an equals sign (`=`).

Adding, Modifying and Deleting Complexes

All **complexes** are administered in a single table. The following commands are used in order to show and modify this

Attribute	Value Specification
`qconf -sc`	Shows a list of all configured complex entries (the complex table).
`qconf -mc`	Opens the editor configured in the $EDITOR environment variable with all configured complex entries. This complex table can be modified then with the editor. When the editor is closed the complex configuration is read in.
`qconf -Mc <filename>`	Reads the given file in as new complex configuration.

Table 67: *Complex* Configuration Commands

Each row of the complex table consists of the following elements:

Attribute	Value Specification
`name`	The unique name of the complex.
`shortcut`	This shortcut can be used instead of the name of complex (e.g. when requesting the complex). It must be unique in the complex configuration.
`type`	The type of the complex variable (used for internal compare functions and load scaling). Possible values are `INT`, `DOUBLE`, `TIME`, `MEMORY`, `BOOL`, `STRING`, `CSTRING`, `RESTRING`, `HOST` and `RSMAP`. See man `complex` for more detailed format descriptions. A `CSTRING` is a case insensitive string type. A `RSMAP` (resource map) is similar to a `INT` but must be a consumable and have additional functionalities, like mapping a job not only to a number of resources but also to specific resource instances.
`relop`	Specifies the relation operator used for comparison of the user requested value and the current value of the complex. The following operators are allowed: `##`, `<`, `>`, `<=`, `>=`, and `EXCL`. The `EXCL` operator allows to define host exclusive of queue exclusive access control.
`requestable`	Possible values are `y`, `yes`, `n`, `no`, and `f`, `forced`. **Yes** means that a user can request this resource, **no** denotes the resource non-requestable, and **forced** rejects all jobs, which do not request this resource.
`consumable`	Possible values are `y`, `yes`, `n`, `no`, and `j`, `job`. Declares a complex as a consumable in case of **yes**. When **job** is set, the complex is a per job consumable. Since 8.1.3 it exists a `h`, `host` consumable which is a per host consumable. This type is only allowed with the type RSMAP.
`default`	When the complex is a consumable, a default request can be set here. It is overridden when a job requests this complex on command line.
`urgency`	Defines the resource urgency. When a user requests this resource, this resource urgency is taken into account when the scheduler calculates the priority of the job (see Urgency Policy).
`aapre`	Defines if a consumable resource will be reported as available when a job that consumes such a resource is preempted. For all non-consumable resources it can only be set to NO. For consumables it can be set to YES or NO. The aapre-attribute of the slots complex can only be set to YES. After the installation of UGE all memory based complexes are defined as consumable and aapre is also set to YES. As result preempted jobs will report memory of those jobs as available that are in the preempted (suspended) state.

Attribute	Value Specification
affinity	Defines the resource affinity factor. A value of 0 means that no affinity is configured for the variable, a positive value means that affinity is configured (jobs already running on a host attract other jobs), a negative value means that anti-affinity is configured (jobs already running on a host reject other jobs). See Affinity, Anti-Affinity, Best Fit

Table 68: *Complex* Configuration Attributes

After a default installation, at least the following complexes are available (there are more depending on the product version):

```
> qconf -sc
#name               shortcut   type        relop requestable consumable default  urgency   aapre
#-------------------------------------------------------------------------------------------------
arch                a          RESTRING    ##    YES         NO         NONE     0         NO
calendar            c          RESTRING    ##    YES         NO         NONE     0         NO
cpu                 cpu        DOUBLE      >=    YES         NO         0        0         NO
display_win_gui     dwg        BOOL        ##    YES         NO         0        0         NO
docker              dock       BOOL        ==    YES         NO         0        0         NO
docker_images       dockimg    RESTRING    ==    YES         NO         NONE     0         NO
h_core              h_core     MEMORY      <=    YES         NO         0        0         NO
h_cpu               h_cpu      TIME        <=    YES         NO         0:0:0    0         NO
h_data              h_data     MEMORY      <=    YES         NO         0        0         NO
h_fsize             h_fsize    MEMORY      <=    YES         NO         0        0         NO
h_rss               h_rss      MEMORY      <=    YES         NO         0        0         NO
h_rt                h_rt       TIME        <=    YES         NO         0:0:0    0         NO
h_stack             h_stack    MEMORY      <=    YES         NO         0        0         NO
h_vmem              h_vmem     MEMORY      <=    YES         NO         0        0         NO
hostname            h          HOST        ##    YES         NO         NONE     0         NO
load_avg            la         DOUBLE      >=    NO          NO         0        0         NO
load_long           ll         DOUBLE      >=    NO          NO         0        0         NO
load_medium         lm         DOUBLE      >=    NO          NO         0        0         NO
load_short          ls         DOUBLE      >=    NO          NO         0        0         NO
m_core              core       INT         <=    YES         NO         0        0         NO
m_socket            socket     INT         <=    YES         NO         0        0         NO
m_thread            thread     INT         <=    YES         NO         0        0         NO
m_topology          topo       RESTRING    ##    YES         NO         NONE     0         NO
m_topology_inuse    utopo      RESTRING    ##    YES         NO         NONE     0          NO
mem_free            mf         MEMORY      <=    YES         NO         0        0         NO
mem_total           mt         MEMORY      <=    YES         NO         0        0         NO
mem_used            mu         MEMORY      >=    YES         NO         0        0         NO
min_cpu_interval    mci        TIME        <=    NO          NO         0:0:0    0         NO
np_load_avg         nla        DOUBLE      >=    NO          NO         0        0         NO
np_load_long        nll        DOUBLE      >=    NO          NO         0        0         NO
```

np_load_medium	nlm	DOUBLE	>=	NO	NO	0	0	NO
np_load_short	nls	DOUBLE	>=	NO	NO	0	0	NO
num_proc	p	INT	##	YES	NO	0	0	NO
qname	q	RESTRING	##	YES	NO	NONE	0	NO
rerun	re	BOOL	##	NO	NO	0	0	NO
s_core	s_core	MEMORY	<=	YES	NO	0	0	NO
s_cpu	s_cpu	TIME	<=	YES	NO	0:0:0	0	NO
s_data	s_data	MEMORY	<=	YES	NO	0	0	NO
s_fsize	s_fsize	MEMORY	<=	YES	NO	0	0	NO
s_rss	s_rss	MEMORY	<=	YES	NO	0	0	NO
s_rt	s_rt	TIME	<=	YES	NO	0:0:0	0	NO
s_stack	s_stack	MEMORY	<=	YES	NO	0	0	NO
s_vmem	s_vmem	MEMORY	<=	YES	NO	0	0	NO
seq_no	seq	INT	##	NO	NO	0	0	NO
slots	s	INT	<=	YES	YES	1	1000	YES
swap_free	sf	MEMORY	<=	YES	NO	0	0	NO
swap_rate	sr	MEMORY	>=	YES	NO	0	0	NO
swap_rsvd	srsv	MEMORY	>=	YES	NO	0	0	NO
swap_total	st	MEMORY	<=	YES	NO	0	0	NO
swap_used	su	MEMORY	>=	YES	NO	0	0	NO
tmpdir	tmp	RESTRING	##	NO	NO	NONE	0	NO
virtual_free	vf	MEMORY	<=	YES	NO	0	0	NO
virtual_total	vt	MEMORY	<=	YES	NO	0	0	NO
virtual_used	vu	MEMORY	>=	YES	NO	0	0	NO

Initializing Complexes

After a complex is configured in the complex configuration, it must be initialized with a meaningful value. The initialization can be done on **global**, **host**, or **queue** level. When a complex is initialized on global level, the complex is available on the complete cluster. In case of a consumable, the accounting for the consumable is done cluster-wide. Host level complexes are available after altering the local cluster configuration on the specific host. They are available and accounted for in all queue instances on the host. Queue level complexes are configured for the complete queue but accounted on the host level.

In order to initialize a pre-configured complex as a **global complex**, the `complex_values` attribute in the global host configuration has to be edited. In the following example, a complex with the name `complexname` is initialized with the value `10`.

```
> qconf -me global
...
complex_values  complexname=10
```

Host complexes are configured similarly, but instead of editing the global host configuration, the local host configuration must be changed.

```
> qconf -me hostname
...
complex_values  complexname=10
```

Queue complexes are configured in the queue configuration:

```
> qconf -mq queuename
...
complex_values  complexname=10
```

After setting this, each queue instance (each host on which the queue is configured) has a complex `complexname` with the value 10 defined. If this complex is a consumable, and the queue spans over 5 hosts, then overall 50 units can be consumed (10 per queue instance). Sometimes the complex must be initialized with different values on each queue instance (i.e. here on different hosts). This can be done with the "[" "]" syntax. The following example assigns the complex `complexname` on queue instance `queue1@host1` 10 units, on `queue1@host2` 5 units, and on all other queue instances 20.

```
> qconf -mq queuename
...
complex_values  complexname=20,[host1=complexname=10],[host2=complexname=5]
```

Using Complexes

After adding and initializing a new complex, the value of the complex can be shown either with `qhost` (host level complexes) or with `qstat` (host, queue, and global complexes).

The `qstat -F <complexname>` shows the state of the complex `complexname` on each queue instance. The output of all available complexes can be seen with `qstat -F`.

```
> qstat -F complexname
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@tanqueray                BIPC  0/0/20         0.10     lx-amd64
    qc:complexname=20
---------------------------------------------------------------------------------
all.q@unertl                   BIPC  0/1/10         0.00     lx-amd64
    qc:complexname=50
```

The prefix `qc` implies that the type of the complex is a queue based consumable. Other common prefixes are `qf` (global complex with a fixed value), and `hl` (host complex based on load value). Host specific values can also be seen by `qhost -F`.

The following table lists the semantic of the both prefix letters.

Attribute	Value Specification
`g`	Cluster global based complex
`h`	Host based complex
`q`	Queue (queue-instance) based complex
`l`	Value is based on load report

Attribute	Value Specification
L	Value is based on a load report, which is modified through through the load scaling facility
c	Value is a based on consumable resource facility
f	The value is fixed (non-consumable complex attribute or a fixed resource limit)

Table 69: Meaning of Different Prefixes from Complexes Shown by qstat and qhost

If a complex is requestable (REQUESTABLE equals YES), then a user can request this resource on job submission time as either a hard or a soft request. A default request is a hard request, which means that the job only runs on execution hosts/queue instances where the resource request can be fulfilled. If requesting a resource as a soft request (see qsub man page -soft parameter), then the Univa Grid Engine scheduler tries to dispatch the job with as few soft request violations as possible.

The following example shows how 2 units of the consumable complex complexname are requested:

```
> qsub -l complexname=2 -b y /bin/sleep 120
```

Configuring Load Sensors

By default, the Univa Grid Engine execution daemons report the most common host load values, such as average CPU load, amount of memory, and hardware topology values such as the number of CPU cores. If more site specific resource state values are needed, Univa Grid Engine supports this with the **load sensor** facility. A load sensor can be a self-created executable binary or a load sensor script that must just follow a few simple pre-defined rules. The communication between the execution daemon and the load sensor is done via standard input and standard output of the load sensor.

Load sensors are registered in the global or local cluster configuration (qconf -mconf, load_sensors), in which the execution host specific local cluster configuration overrides the global configuration. Load sensors are registered as a comma separated list of absolute paths to the load sensors.

WIN-only The path to Windows load sensors must be configured in UNIX notation and the path_map file must contain the corresponding mapping. Both Windows batch scripts and Windows executables can be configured.

A correct load sensor must respect the following rules:

- The load sensor must be implemented as an endless loop.
- When “quit” is read from STDIN, the load sensor should terminate.
- When end-of-line is read from STDIN, the load sensor has to compute the load values and write the load sensor report to STDOUT.

The load sensor report must have the following format:

- A report starts with a line containing either the keyword "start" or the keyword "begin".
- A report ends with a line containing the keyword "end".
- In between, the load values are sent. Each load value is a separate line with the following format:

`host:name:value`. The `host` denotes the host on which the load is measured or "global" in the case of a global complex. The `name` denotes the name of the resource (complex) as specified in the complex configuration. The `value` is the load value to be reported.

`WIN-only` In Windows executable load sensors, it's necessary to flush STDOUT after the keyword "end" was written to it. Otherwise, the load values will be transferred to the execution daemon not before the STDOUT buffer is full. In batch load sensors, the "echo" command flushes STDOUT.

Sample load sensor scripts can be found here: `$SGE_ROOT/util/resources/loadsensors/`. Also consider the man page `sge_execd(8)` for additional information.

2.5.4 Configuring and Using the RSMAP Complex Type

This section describes the new complex type RSMAP, which was introduced in Univa Grid Engine version 8.1.0.

When managing host resources like GPU devices it is not always sufficient to handle just the amount of installed resources, which is usually done with an integer host consumable. Furthermore there is a need to orchestrate access to specific devices in order to make sure that there are no conflicts for individual devices and that they can be accessed properly. Traditionally this is done by wrapper scripts, which are installed on the execution hosts or by partitioning hosts through different queues. The new host complex type RSMAP not only restricts the amount of units of a resource used concurrently on a host, it also attaches identifiers to resource units and assigns them to the jobs when they get dispatched to that host. Thereby a job gets attached to a specific unit (an ID) of such a resource.

The first section below shows how such an RSMAP complex is created. Afterwards the usage of such complexes is shown. The last section shows the behavior of the complex types in conjunction with special job types, like parallel jobs and job arrays.

Creating and Initialization of RSMAP Complexes

A new complex of the type RSMAP (**resource map**) is created like all other complexes with the `qconf` command. Usually the complex is added by hand meaning using an editor with `qconf -mc`.

As all complexes it needs a **name** and a **shortcut**. The **type** must be `RSMAP`. The only allowed **RELOP** is `<=`. It should be made **requestable** from command line. It must be a **consumable** (**YES** or in case of a per job consumable **JOB**. Since Univa Grid Engine 8.1.3 it can also be set to **HOST**, which means that the requested amount is handled as a per host request for all hosts the job might span). Since the default values are not attached to jobs (which is very important for the RSMAP), setting a **default** value is not allowed. Depending on the scheduler configuration, the attribute **urgency** can be increased in order to increase the scheduling priority of jobs requesting this complex attribute.

A **RSMAP** maps jobs to one or more specified IDs. An ID can be any kind of number or string and IDs must not (but can) be unique. Those IDs are either cluster global or host specific. With one complex type you can define a set of different IDs. Note that this can also be used to reduce the need for multiple complexes with versions prior to 8.1 in some situations.

Example:

Adding a new RSMAP complex called "ProcDevice" to the Univa Grid Engine system.

```
> qconf -mc
#name     shortcut   type        relop requestable consumable default  urgency
#-------------------------------------------------------------------------------
ProcDevice  pd       RSMAP        <=    YES         YES        0        0
```

After the new complex is announced to the Univa Grid Engine system it is initialized with different IDs. This can be done either on the host level **or** on the cluster global level. It can not be used on the queue level.

Example:

Initializing the "ProcDevice" complex on host `host1` with the IDs "device0" and "device1".

```
> qconf -me host1
  ...
  complex_values    ProcDevice=2(device0 device1)
```

The initialization line is similar to a consumable but with an additional specifier in brackets denoting the ID list. IDs can also be numerical ranges like `(1-10)` which is a shortcut for `(1 2 3 4 5 6 7 8 9 10)`. You can also mix ranges with arbitrary strings (`(NULL 100-113 INF)`), and also multiple identical values are allowed `(0 0 0 0 1 1 1 1)`. Those can be useful, for instance, if devices are installed, which can be accessed multiple times.

Usage of RSMAP Complexes

Like consumables, the resource map values can be requested during job submission. The following example shows two jobs requesting each one of the configured "ProcDevices" (in the examples above).

```
> qsub -b y -l ProcDevice=1 sleep 3600
Your job 9 ("sleep") has been submitted
> qsub -b y -l ProcDevice=1 sleep 3600
Your job 10 ("sleep") has been submitted
...
> qstat -j 9
==============================================================
job_number:                 9
...
hard resource_list:         ProcDevice=1
...
**resource map    1:          ProcDevice=macsuse=(device0)**
...
```

```
> qstat -j 10
==============================================================
job_number:                 10
...
hard resource_list:         ProcDevice=1
...
**resource map    1:          ProcDevice=macsuse=(device1)**
...
```

As you can see, each job has received a different device ID. This actual device ID can be accessed via the `SGE_HGR_ProcDevice` environment variable.

Example:

The device ID is needed by an application as parameter. The application is started by a job scripts.

```
#!/bin/sh
...
myapp $SGE_HGR_ProcDevice
...
```

RSMAP Topology Masks

In certain situations it is useful to map specific host resources to specific host topology entities, like CPU cores and sockets. In case of an multi-socket NUMA machine, some resources (like Co-processors or network devices) are connected to specific sockets. Univa Grid Engine supports such configurations by allowing to define topology masks in the RSMAP configuration.

RSMAP topology masks are strings, very similar to topology strings (complex values m_topology and m_topology_inuse), which prohibit that jobs using certain masked CPU cores (or complete sockets). A topology string for an eight core processor for example is "SCCCCCCCC", while a two socket quad-core system is represented by "SCCCCSCCCC". Allowed characters in a topology mask are "S" ("s"), which represents a socket, "C" which is an allowed (not masked core), "c", which is a masked (not allowed) core, and "T" ("t"), which represents a hardware thread. Since single threads can not be masked (only complete cores), the "T"/"t" character is ignored. Important are the "C" and "c" values, which allow or disallow the usage of the cores.

Examples: The topology mask "SccSCC" allows the usage of the second socket (with core 0 and 1), while the usage of the first socket is prohibited. The topology mask "SCcCCSccCC" allows the usage of first, third, and fourth core on the first socket and the third and fourth core of the second socket.

In order to map a host resource with a certain topology, such a topology mask must be appended in the RSMAP initialization. In the example of the last subsection a resource map containing 2 devices was defined.

```
complex_values    ProcDevice=2(device0 device1)
```

Let's assume that `device0` is connected to the first socket, and `device1` to the second socket of a dual-socket quad-core machine. To make the Univa Grid Engine scheduler aware of this mapping, a topology mask must be appended to the configuration. First check that in the complex configuration in the consumable column, the value `HOST` is set. RSMAPs with topology masks must be `HOST` consumables since CPUs are host resources and have to be chosen independent from any queues a parallel job might run in. After this is done the topology mask can be set by appending it with an additional colon.

```
qconf -me <hostname>
...
complex_values    ProcDevice=2(device0:SCCCCScccc device1:ScccSCCCC)
```

When the scheduler selects the `device0` for the job, the job is automatically bound to the free cores of the first socket (because the topology mask marks the second socket internally as being used). If there is already a job running, which is bound on cores of the first socket, only the unbound cores can be used for the job. When the job comes with an own (implicit or explicit) core binding request, that request is honor as well. But it is never possible that the job gets bound to cores of the second socket.

Two examples demonstrating this behavior:

First submit a job requesting one core:

```
$ qsub -b y -binding linear:1 -l ProcDevice=1 sleep 123
Your job 29 ("sleep") has been submitted

$ qstat -j 29
 ==============================================================
...
binding               1:    maui=0,0
resource map          1:    ProcDevice=maui=(device0)
```

The jobs gets one core on the selected device, allowed from the topology mask.

Now submit a job requesting just the device.

```
$ qsub -b y -l ProcDevice=1 sleep 123
Your job 30 ("sleep") has been submitted

$ qstat -j 30
==============================================================
binding               1:    maui=1,0:1,1:1,2:1,3
resource map          1:    ProcDevice=maui=(device1)
```

The job gets all free cores, which are allowed by the RSMAP topology mask.

Special Jobs

The resource map (RSMAP) complex can be used in conjunction with other job types, like array jobs and parallel jobs, as well. The next sections are describing the behavior and are showing some detailed examples.

Array jobs and the RSMAP Complex

When an array job is submitted then the same job (job script or binary) is started multiple times. This can be useful for data parallel applications, where the same processing must be performed on different data sets. Like a job, each of the array job task can be assigned to a resource ID out of the created pool (ID list). During job submission the resources requests are done per task, hence when requesting one resource ID but two tasks, then each task gets a different resource ID (if they are unused).

Example:

In this example we configure 10 new devices on host `host1` with device numbers 100, 101, ..., 109.

```
> qconf -mc
#name      shortcut   type  relop requestable consumable default  urgency
#---------------------------------------------------------------------------
devices     dev      RSMAP     <=    YES          YES        0        0

> qconf -me host1
...
complex_values        dev=10(100-109)
...
```

Now an array job with 10 tasks is started, which requests 2 devices per task.

```
> qsub -l devices=2 -t 1:10 $SGE_ROOT/examples/jobs/sleeper.sh 3600
Your job-array 5.1-10:1 ("Sleeper") has been submitted

> qstat
job-ID  prior   name   user  state submit/start at     queue    slots ja-task-ID
-----------------------------------------------------------------------------
 5 0.55500 Sleeper     user    r   07/07/2011 09:04:56 all.q@host1     1 1
 5 0.55500 Sleeper     user    r   07/07/2011 09:04:56 all.q@host1     1 2
 5 0.55500 Sleeper     user    r   07/07/2011 09:04:56 all.q@host1     1 3
 5 0.55500 Sleeper     user    r   07/07/2011 09:04:56 all.q@host1     1 4
 5 0.55500 Sleeper     user    r   07/07/2011 09:04:56 all.q@host1     1 5
 5 0.55500 Sleeper     user    qw  07/07/2011 09:04:55                 1 6-10:1
```

As you can see just 5 jobs are able to run because the devices are limited to 10 at a time. All jobs are running on `host1`.

```
> qstat -j 5
==============================================================
job_number:                 5
...
hard resource_list:         devices=2
...
**resource map    1:          devices=host1=(100 101)**
```

```
**resource map    2:          devices=host1=(102 103)**
**resource map    3:          devices=host1=(104 105)**
**resource map    4:          devices=host1=(106 107)**
**resource map    5:          devices=host1=(108 109)**
scheduling info:            (-l devices=2) cannot run in queue "all.q@host3"
                                because job requests unknown resource  (devices)
                                (-l devices=2) cannot run in queue "all.q@host2"
                                because job requests unknown resource  (devices)
                                (-l devices=2) cannot run at host "host1" because
                                it offers only hc:devices=0.000000
```

The `qstat` output shows that each task has been allocated two different IDs.

Parallel jobs and the RSMAP Complex

Resource maps can also be used by parallel jobs. The amount of IDs a parallel job is granted is exactly the same like for traditional consumables. If a parallel job with, for instance, 4 slots has an additional request for a resource map complex, then the total granted IDs are **slots * requested amount**. Sometimes the amount of resources needed by the job does not depend on the amount of granted slots. Then a resource map (RSMAP) complex can be configured to allocate the amount of requested resources during submission time (without multiplication by slots). This is accomplished by setting **job** instead of **YES** for the **consumable** attribute in the complex configuration (`qconf -mc`). Then the resource map is a **per job** consumable complex. The drawback is that a per job resources are only granted on the host of the master task. Hence RSMAP also allows to be configured as **per host** consumables, when the **consumable** attribute is set to **HOST**. That allows to request a specific amount of resources per host independent from the amount of granted slots per host.

Example:

In this example we are submitting an OpenMP job, which runs on exactly one host but running a specific amount of threads performing work in parallel. This job needs access to one GPU regardless how many OpenMP threads are spawned.

In order to declare the GPU devices (lets say 2 are added on host1), the complex must be created. Because a fixed number of GPUs are accessed per job (and not per granted slot) the complex is a **JOB** consumable.

```
> qconf -mc
#name    shortcut   type    relop requestable consumable default  urgency
#--------------------------------------------------------------------------
GPGPU   gp         RSMAP     <=    YES         **JOB**        0        0
```

Now the complex must be initialized on `host1`.

```
> qconf -me host1
...
complex_values      GPGPU=2(GPU0 GPU1)
...
```

The parallel environment (here called **mytestpe**) must be setup for the OpenMP jobs with `$pe_slots`.

```
> qconf -mpe mytestpe
pe_name            mytestpe
slots              24
user_lists         NONE
xuser_lists        NONE
start_proc_args    NONE
stop_proc_args     NONE
allocation_rule    $pe_slots
control_slaves     FALSE
job_is_first_task  TRUE
urgency_slots      min
accounting_summary FALSE
```

Now start the parallel job using 12 cores but just one GPU.

```
> qsub -pe mytestpe 12 -l GPGPU=1 jobscript

> qstat -j 6
==============================================================
job_number:                 6
...
hard resource_list:         GPGPU=1
...
parallel environment:  mytestpe range: 12
binding:                    NONE
usage           1:          cpu=00:00:00, mem=0.00044 GBs,
                            io=0.00012, vmem=19.125M, maxvmem=19.125M
binding         1:          NONE
resource map    1:          GPGPU=macsuse=(GPU0)
```

The parallel job has received just one GPU device because the complex was configured as a per job resource map. If it would be a consumable (**YES**) it couldn't be started because 12 IDs would be needed and would have to be configured for the complex attribute GP.

2.5.5 Advanced Attribute Configuration

With Univa Grid Engine, it is also possible to modify internal objects directly. The following table shows the commands supporting direct object configurations:

Attribute	Value Specification
`qconf -aattr obj_nm attr_nm val obj_id_list`	Adds a new specification of an attribute/value pair into an object (queue, exechost, hostgroup, pe, rqs, ckpt) with a specific characteristic (e.g. in case of an queue, it is added only to the queue with the name defined in `obj\_id\_list`).
`qconf -Aattr obj_nm fname obj_id_list`	Same as above but the attribute name and attribute value is taken from a file given by the file name `fname`.

Attribute	Value Specification
`qconf -dattr obj_nm attr_nm val obj_id_list`	Deletes an object attribute.
`qconf -Dattr obj_nm fname obj_id_list`	Deletes an object attribute by a given file.
`qconf -mattr obj_nm attr_nm val obj_id_list`	Modifies an object attribute.
`qconf -Mattr obj_nm fname obj_id_list`	Modifies an object attribute based on a given file.
`qconf -purge obj_nm3 attr_nm objectname`	Removes overriding settings for a queue domain (queue@@hostgroup) or a queue instance. If a hostgroup is specified, it just deletes the settings for the hostgroup and not for each single queue instance.
`qconf -rattr obj_nm attr_nm val obj_id_list`	Replaces an object attribute.
`qconf -Rattr obj_nm fname obj_id_list`	Replaces an object attribute based on a given file.

TABLE: Commands for Direct Object Modification

Example: Modification of a Queue Configuration

The following example shows how this direct object attribute modification can be used to initialize a queue consumable.

First add a new consumable `test` to the complex configuration.

```
> qconf -sc > complexes; echo "test t INT <= YES NO 0 0" >> complexes;
qconf -Mc complexes user@host added "test" to complex entry list
```

Show the default complex initialization of queue `all.q`.

```
> qconf -sq all.q | grep complex_values
complex_values        NONE
```

Now initialized the consumable `test` with the value `2` on all queue instances defined by the queue `all.q`.

```
> qconf -aattr queue complex_values test=2 all.q
user@host modified "all.q" in cluster queue list
```

Show the initialization:

```
> qconf -sq all.q | grep complex_values
complex_values        test=2
```

Now add a different initialization value for the queue instance `all.q@macsuse`.

```
> qconf -aattr queue complex_values test=4 all.q@macsuse
user@host modified "all.q" in cluster queue list
```

Show the updated queue attribute.

```
> qconf -sq all.q | grep complex_values
complex_values        test=2,[macsuse=test=4]
```

Remove the configuration for `all.q@macsuse`.

```
> qconf -purge queue complex_values all.q@macsuse
user@host modified "all.q" in cluster queue list
```

Now show the queue configuration again:

```
> qconf -sq all.q | grep complex_values
complex_values        test=2
```

2.5.6 Configuring and Using Linux cgroups

Newer Linux kernels and distributions support a facility called **control groups** (cgroups) for improved resource management. Univa Grid Engine has added support for cgroups in version 8.2 for *lx-amd64* hosts. The following requirements must be fulfilled for the features to work correctly:

- Linux kernel must have support for cgroups (e.g. *RHEL 6.0* or later)
- Some distributions require the cgroup package to be installed.
- cgroups subsystems (memory, cpuset, freezer) need to be mounted to different directories with the subsystem as name of the directory (like */cgroup/memory*).
- All subsystems must be mounted in the same parent directory (like */cgroup*), this directory is called here cgroup path.

The memory subsystem needs to have the following configuration parameters available:

`_memory.limit_in_bytes_, _memory.soft_limit_in_bytes_, _memory.memsw.limit_in_bytes_.`

The cpuset subsystem needs to support: *cpuset.cpus*, *cpuset.mems*.

Availability can be checked by doing an *ls* on the subsystem (like *ls /cgroup/memory*). Please consult your OS documentation for activating / mounting cgroups. Univa Grid Engine also supports trying to auto-mount subsystems, but this might fail depending on the Linux distribution / version.

Tested Linux distributions at the time of initial release of the feature: *RHEL / CentOS >= 6.0-6.4. Mint Linux 14. Ubuntu 12.04. openSUSE 12.3.*

Further details of the cgroups implementation can be found in the *sge_conf* manual page.

Enabling cgroups support in Univa Grid Engine

Since support for cgroups depends on the host type it can be configured in the global (*qconf -mconf global*) and / or local host configuration (*qconf -mconf*) in a new attribute called *cgroups_params*. In the default configuration after installation it is turned off:

```
qconf -sconf global
cgroups_params            cgroup_path=none subdir_name=UGE \
                          cpuset=true mount=false freezer=false \
                          freeze_pe_tasks=false killing=false \
                          forced_numa=false h_vmem_limit=false \
                          m_mem_free_hard=false m_mem_free_soft=false \
                          min_memory_limit=0
```

When **cgroups_params** are configured in the local host configuration they override the global configuration parameters. If a parameter is not listed in the local configuration then the value of the global parameter is used. If the parameter is not listed anywhere then the default value (none, false, or 0) is used.

In order to enable cgroups the cgroup_path must be set to the correct (existing) path. Typical locations are /cgroup or /sys/fs/cgroup. If **cgroup_path** is set to none then cgroups is disabled, even if other parameters are set.

If **mount** is set to 1 or true then Univa Grid Engine tries to mount a subsystem if it is not mounted yet.

In the following subsections the different behaviour of jobs running within cgroups are explained.

Enabling cpuset for core binding

When using core binding in Linux system calls are made (like *sched_setaffinity*) before the jobs start in order to bind it on Univa Grid Engine selected processor ids. One drawback is that the user application can revert that by calling the same system calls. When the cgroup parameter **cpuset** is set to 1 or true Univa Grid Engine puts the job into a cpuset cgroup with the specific CPU ids assigned. This ensures that the job cannot re-bind the job to other than granted CPUs.

If **forced_numa** is set to 1 or true then on NUMA machines only local memory (memory in the same NUMA zone) is allowed to be used when the job requested memory allocation with *-mbind cores:strict*.

Using cgroups for job suspend and resume

The default suspension mechanism is sending SIGSTOP signals to the application and SIGCONT for resumption. This is not always what the application needs because SIGCONT is catchable and may break functionality of the application. The cgroups *freezer subsystem* allows suspend and resume without sending signals. The scheduler just does not schedule the process anymore, it stays in the same state like waiting for I/O resources (P state). If the **freezer** parameter is set to 1 or true than the whole job (without the shepherd process) is frozen by the kernel. This behavior can made application specific by overriding it in the queue configuration. If a job needs

the signal then in the queue where the job runs the *suspend_method* can be set to SIGSTOP and resume to SIGCONT. For those jobs the freezer subsystem is turned off.

The behavior for tightly integrated parallel jobs can be controlled by the **freeze_pe_tasks** parameter. If set to false (default) then slave tasks are not put in the freezer (also slaves on master host which are started by a new shepherd using *qrsh*). If set to 1 or true than all slave tasks are frozen (also slaves on remote hosts). If a queue overrides the freezer by a signal then the **execd_param** *SUSPEND_PE_TASKS* is taken into account (true when not set) for the appropriate behavior.

Using cgroups for main memory and swap space limitation

Main memory can be limited by using the cgroups *memory subsystem*. It is controlled by the cgroups parameters **m_mem_free_hard** and **m_mem_free_soft**. If **m_mem_free_hard** is set to 1 or true then the Linux kernel ensures that the job does not use more main memory than required. For main memory footprint also the shepherd process is taken into account. The limit can be requested with the **m_mem_free** memory request.

```
qsub -l m_mem_free=12G ...
```

For parallel jobs the memory request is multiplied by the amount of slots the job got granted on the specific host. Note that some Linux distributions (e.g. RHEL) have bugs in counting memory for forked processes so that the overhead could be very high (200M instead of a few megabyte per shepherd). This can be tested by looking at the h_vmem values in the accounting file (qacct -j). If **m_mem_free_soft** is set to 1 or true (and hard memory limit is turned off) then the requested memory with *m_mem_free* is a soft limit. Depending on the Linux kernel it allows the process to consume more than allowed memory if there is still free memory left on the execution host. If the memory is too low the jobs memory footprint is pushed back to the limits (consuming more swap space). More details can be found in the Linux kernel documentation.

Main memory and swap space can be limited with cgroups by setting **h_vmem** to 1 or true. Then instead of setting *rlimits* the job is put into a cgroup using the memory subsystem setting the *memory.memsw.limit_in_bytes* parameter. Together with a lower *m_mem_free* request this allows to enforce main memory limit for the job when the host is at its limits. This eliminates the risk that a job slows down other jobs when consuming more than its requested memory. The job will slow down itself or fail.

A host based minimum memory limit can be set by the **min_memory_limit** parameter which accepts Univa Grid Engine memory values (like bytes or values like 10M, 1G). If a job requested a memory limit and the limit is smaller than this configured value the memory limit is automatically increased to the limit without affecting job accounting. This is useful to solve host related issues of too high memory footprints of jobs or to prevent that users set too low limits for their jobs.

cgroups based killing

Under some rare conditions processes of a job can survive Univa Grid Engine induced job termination. This risk can be eliminated by using the killing parameter. If set to 1 or true Univa Grid Engine signals all processes forked / started by the job until all of them are killed.

Examples

In the following examples first the configuration of **cgroups_params** is shown and afterwards the pertinent command line requests for submitting corresponding jobs.

Restricting main memory

Enabling hard memory limitation for host *oahu*.

```
$ qconf -sconf oahu > oahu
$ echo "cgroups_params cgroup_path=/sys/fs/cgroup m_mem_free_hard=true" >> oahu
$ qconf -Mconf oahu
daniel@oahu modified "oahu" in configuration list
$ qconf -sconf oahu
#oahu:
xterm                         /usr/bin/xterm
mailer                        /bin/mail
cgroups_params                cgroup_path=/sys/fs/cgroup m_mem_free_hard=true
```

Submitting a job requesting 768M main memory. Note that m_mem_free on the host must be lower than the requested value (check *qhost -F -h oahu* for this).

```
$ qsub -b y -l h=oahu,m_mem_free=768M memhog -r100 512M
```

Checking the actual limit on host by inspecting cgroups configuration (when the job is running).

```
$ cat /sys/fs/cgroup/memory/UGE/151.1/memory.limit_in_bytes
805306368
```

An interactive job can be submitted in the same way:

```
$ qrsh -l m_mem_free=4G
```

On the remote host the qrsh session is limited to 4G of main memory. In the following remote session 1G main memory is requested by the memhog utility. Afterwards 3G of main memory are occupied by the next start of memhog. This leads to an abortion of the memhog process but the *qrsh session* is unaffected by this. Hence the next call of memhog requesting 1G is again successful.

```
$ memhog -r1 1G
$ ...
$ memhog -r1 3G
$ ...Killed
$ memhog -r1 1G
$ ...
```

Job suspend and resume by using the freezer subsystem

This is an example configuration of a homogeneous cluster, where all hosts have the same configuration. The configuration is only done once in the *global* host object. All cgroup subsystems are turned on and the minimum memory limit for each job is set to 100M.

```
$ qconf -sconf global
...
cgroups_params               cgroup_path=/sys/fs/cgroup cpuset=true mount=true \
                             freezer=true freeze_pe_tasks=true killing=true \
                             forced_numa=true h_vmem_limit=true \
                             m_mem_free_hard=true m_mem_free_soft=true \
                             min_memory_limit=100M
```

Job suspension can be triggered in different ways: suspend on subordinate, queue based suspension, or manual suspension.

The following job with number 152 is suspended manually now using the freezer subsystem.

```
$ qmod -sj 152
```

When calling qstat the job is in **s** state. On the execution host the job is put in the freezer cgroup.

```
$ cat /sys/fs/cgroup/freezer/UGE/152.1/freezer.state
FROZEN
```

After resume it is in the normal state again.

```
$ qmod -usj 152
daniel - unsuspended job 152
$ cat /sys/fs/cgroup/freezer/UGE/152.1/freezer.state
THAWED
```

Restricting main memory and swap space usage

The cgroups configuration required for this setup is:

```
$ qconf -sconf oahu
...
cgroups_params   cgroup_path=/sys/fs/cgroup m_mem_free_soft=true h_vmem_limit=true
```

Submitting a job requesting 512M main memory and 1024M main memory plus swap space.

```
$ qsub -b y -l h=oahu,m_mem_free=512M,h_vmem=1024M <yourjob>
```

The current kernel behavior is that the job keeps running but the kernel enforces that the main memory usage is pushed back to 512M while the remaining memory is in swap space, in case the host memory is low.

2.6 Monitoring and Modifying User Jobs

Refer to section 'User Guide -> Monitoring_and_Controlling_Jobs - Monitoring and Controlling Jobs in the User's Guide' for information on how to monitor jobs and how to use Univa Grid Engine commands to make modifications to waiting or already executing jobs.

In addition to the modifications a user can do, an administrator can also do the following:

- Monitor and modify jobs of all users
- Set the scheduling priority of a job to a value above the default of 0. The administrator may set this priority to values between -1023 and 1024. This is done with the "-p priority" option of `qalter`.
- Force the immediate deletion of a job in any case. As a normal user, the "-f" option of `qdel` can be used only if ENABLE_FORCE_QDEL is specified in the `qmaster_params` setting of the cluster global configuration. Even if this is specified, the normal user still can't force the immediate deletion of a job; the job will first be deleted in the normal way, and only if this fails will the deletion be forced. As an administrator, the job deletion will immediately be forced.

2.7 Diagnostics and Debugging

The sections below describe aspects of diagnosing scheduling behavior and obtaining debugging information.

2.7.1 KEEP_ACTIVE functionality

Usually, as soon as a job finishes (it does not matter if it finishes successfully or with an error) the jobs-directory gets deleted immediately. With the KEEP_ACTIVE parameter it is possible to customize this behavior. The **KEEP_ACTIVE** switch is an execution daemon parameter (execd_params) which can be set via qconf -mconf.

Parameter	Description
FALSE	If set to false, the job-directory will be deleted after the job finishes.
TRUE	If set to true, the job-directory gets not deleted and stays at the execution host.
ERROR	If set to error, on job error the job-directory will be sent to the qmaster before it gets deleted from the execd-host.
ALWAYS	If set to always, every job-directory will be sent to the qmaster before it gets deleted from the execd-host.

Table 71: KEEP_ACTIVE Parameters

In case of ERROR and ALWAYS, qmaster will copy beside the job-directory also the generated job-script, all job-related execd-messages as also a summary of the `$TMPDIR` of the job to

`$SGE_ROOT/$SGE_CELL/faulty_jobs/$job_id`.

Structure of the jobs-directory

The active jobs directory (`$execd_spool_dir/$hostname/active_jobs/$job_id.$task_id/`) and contains the following files:

File	Description
addgrpid	Additional group-ID
config	Config of the job
environment	List of environment variables which are available in the job shell
error	Error messages of the shepherd
exit_status	Exit status of the job script
job_pid	PID of the job on the execution host
pe_hostfile	Contains all hosts where the parallel job is running
pid	PID of the corresponding shepherd on the execution host
trace	Messages file of the corresponding shepherd
.	Directory which includes all files listed above for every pe-task (only in case of tightly integrated parallel jobs).

2.7.2 Diagnosing Scheduling Behavior

Univa Grid Engine provides several means that help clarify why the scheduler makes specific decisions or what decisions it would make based on the current cluster state for a job with specific requirements.

The `qselect` command prints the list of possible queues to which a job with the given requirements could be scheduled. `qselect` options are listed below:

- specify all requirements the job has using the "-l" option
- limit the possible queues using the "-q" and "-qs" option
- specify the job user with the "-U" option
- specify the available parallel environments using the "-pe" option

The `-w p` option specified in the `qsub, qsh, qrsh` or `qlogin` command line prints the scheduler decisions that would be made for this job with the current cluster state, but does not submit the job. When specified in the `qalter` command line, the `-w p` option prints this list for a job that is queued and waiting. This is a rather efficient way to get the scheduling info, but it provides the data only for this very moment.

`qstat -j <job_id>` prints the "scheduling_info:" for the given job. This is the same data that `qalter -w p <job_id>` prints, except that it is collected for the whole lifetime of the job. This information is available only if the "schedd_job_info" configuration value is set to true in the

scheduler configuration. Note that having "schedd_job_info" set to true may have severe impacts on the scheduler performance.

`qconf -tsm` triggers a scheduler run and writes data for all currently queued jobs to the file `$SGE_ROOT/$SGE_CELL/common/schedd_runlog`. This slows down the scheduler run significantly, but is done only for this one scheduler run.

By setting the "params" configuration value to "MONITOR=1" in the scheduler configuration, the scheduler writes one or more lines for every decision it makes about a job or a task to the file `$SGE_ROOT/$SGE_CELL/common/`schedule. This is described in detail below in the section "Turning on Debugging Information"/"Activating Scheduler Monitoring". This option also slows down the scheduling process.

Scheduler profiling helps answer the question of why a scheduler run might be taking so long. Enable scheduler profiling by setting the "params" configuration value to "PROFILE=1" in the scheduler configuration. The scheduler then writes statistics about the scheduler run times to the Qmaster messages file. This is described in detail below in the section "Turning on Debugging Information"/"Activating Scheduler Profiling".

2.7.3 Location of Logfiles and Interpreting Them

The daemons of Univa Grid Engine write their status information, warnings and errors to log files, as follows.

Daemon	Log file
sge_qmaster	<sge_qmaster_spool_dir>/messages
sge_shadowd	<sge_qmaster_spool_dir>/messages_shadowd.
sge_execd	<sge_execd_spool_dir>/messages
sge_shepherd	<sge_execd_spool_dir>/active_jobs/<job_dir>/trace
sge_container_shepherd <	sge_execd_spool_dir>/active_jobs/<job_dir>/container_trace
dbwriter	$SGE_ROOT/$SGE_CELL/common/spool/dbwriter/dbwriter.log

Table 73: Daemon Log File Locations

- <sge_qmaster_spool_dir> is the "qmaster_spool_dir" that is defined in the `$SGE_ROOT/$SGE_CELL/common/` bootstrap file.
- <host> is the name of the host on which the sge_shadowd is running.
- <sge_execd_spool_dir> is the "execd_spool_dir" from the global or the host local configuration ("qconf -sconf" resp. "qconf -sconf ").
- <job_dir> is composed from the job ID and the task ID, e.g. "42.1".

All "messages" and "messages_shadowd.<host>" files have the same structure:

```
05/20/2011 14:27:49|  main|kailua|I|starting up UGE 8.0.0 (lx-x86)
05/20/2011 14:30:07|worker|kailua|W|Change of "execd_spool_dir" will not be
               effective before sge_execd restart as described in sge_conf(5)
05/20/2011 14:30:23|worker|kailua|E|There are no jobs registered
05/20/2011 14:30:24|worker|kailua|E|sharetree does not exist
05/20/2011 14:30:47|worker|kailua|I|using "/var/spool/gridengine/4080/execd"
               for execd_spool_dir
05/20/2011 14:30:47|worker|kailua|I|using "/bin/mail" for mailer
```

The columns contain the date, time, thread name, host name, message type and the message itself.

- Date, time and the host name describe when and where the line was written to the log file.
- The thread name is always “main”, except for the sge_qmaster which has several threads.
- The message type is one of C(ritical), E(rror), W(arning), I(nfo) or D(ebug). Which messages are logged is controlled by the “loglevel” setting in the global configuration. If this is set to “log_error”, only messages of type “C” and “E” are logged; if it is “log_warning”, additionally the messages of type “W” are logged; for “log_info” messages of type “I” are also logged; and for “log_debug” messages of all types are logged.

The “trace” file of the shepherd and the is available only while the job is running, except when the “execd_params” “KEEP_ACTIVE=TRUE” is set; then it is also available after the job ends. The same applies to the “container_trace” file of the “sge_container_shepherd”, which is started only for non-autostarting Docker jobs.

Such a trace file looks like this:

```
05/23/2011 15:09:00 [1000:26811]: shepherd called with uid = 0, euid = 1000
05/23/2011 15:09:00 [1000:26811]: starting up 8.0.0
05/23/2011 15:09:00 [1000:26811]: setpgid(26811, 26811) returned 0
05/23/2011 15:09:00 [1000:26811]: do_core_binding: "binding" parameter not
                                  found in config file
```

The columns contain the date and time, the effective user ID and the process ID of the sge_shepherd process and the message itself.

The log file of the dbwriter looks like this:

```
23/05/2011 14:14:18|kailua|.ReportingDBWriter.initLogging|I|
       Starting up dbwriter (Version 8.0.0)
---------------------------
23/05/2011 14:14:18|kailua|r.ReportingDBWriter.initialize|I|
       Connection to db jdbc:postgresql://kailua:5432/arco
23/05/2011 14:14:19|kailua|r.ReportingDBWriter.initialize|I|
       Found database model version 10
23/05/2011 14:14:19|kailua|tingDBWriter.getDbWriterConfig|I|
       calculation file /gridengine/dbwriter/database/postgres/dbwriter.xml
       has changed, reread it
```

```
23/05/2011 14:14:19|kailua|Writer$VacuumAnalyzeThread.run|I|
       Next vacuum analyze will be executed at 24.05.11 12:11
23/05/2011 14:14:19|kailua|ngDBWriter$StatisticThread.run|I|
       Next statistic calculation will be done at 23.05.11 15:14
23/05/2011 14:15:19|kailua|er.file.FileParser.processFile|I|
       Renaming reporting  to reporting.processing
23/05/2011 14:15:19|kailua|iter.file.FileParser.parseFile|W|
       0 lines marked as erroneous, these will be skipped
23/05/2011 14:15:19|kailua|iter.file.FileParser.parseFile|I|
       Deleting file reporting.processing
```

Here again, the first two columns are date and time, then the name of the host on which the dbwriter is running, the right-most part of the function that did the logging, the type of the message and the message itself.

If the particular module of Univa Grid Engine can't write to the configured directory for the messages file or it does not get the configuration value at all or it cannot create that directory, it writes a panic file to the `/tmp` directory (`c:\tmp` directory on Windows). These are the paths of the panic files:

Daemon	Log file
sge_qmaster	/tmp/qmaster_messages.<pid>
sge_shadowd	/tmp/shadowd_messages.<pid>
sge_execd (Unix)	/tmp/execd_messages.<pid>
sge_execd (Windows)	C:\tmp\execd_messages.<pid>
sge_shepherd (Unix)	/tmp/shepherd.<pid>
sge_shepherd (Windows)	C:\tmp\shepherd.<pid>
sge_container_shepherd	/tmp/container_shepherd.<pid>

Table 74: Daemon Panic File Locations

If the particular module of Univa Grid Engine can't create or open its messages file, but can create a file in the configured directory, it creates a file called "messages.<pid>.<id>", where the "id" is increased if the next message cannot be written to that file again.

2.7.4 Turning on Debugging Information

The debugging sections describe recommended debugging tools available in Univa Grid Engine, including scheduler profiling and log-files.

Activating Scheduler Profiling

The Univa Grid Engine profiling functionality is used during the development of the software to analyze the performance of the scheduler component. Also in customer environments the profiling can be used to detect issues in the setup of the cluster.

With the profiling module enabled in the scheduler component, profiling is running as a thread within the `sge_qmaster` process and will print additional log messages to the message file of the master component. The message file can be found in the directory `$SGE_ROOT/$SGE_CELL/spool/qmaster/`

Each line in the output is introduced by the following:

- time when the output was made,
- the name of the thread that caused the logging,
- the hostname on which the component is running,
- a letter that shows what kind of logging message was printed (*P* for profiling)
- and the logging message itself:

05/13/2011 08:42:07|schedu|host1|P|PROF: ...

The line above shows profiling output (*P*) of the *scheduler* thread that was running on host *host1*. Profiling messages themselves will either start with *PROF:* or *PROF():*.

For simplicity, the prefixed text of each line has been skipped in the following sample output:

```
01    PROF: sge_mirror processed 5 events in 0.000 s
02    PROF: static urgency took 0.000 s
03    PROF: job ticket calculation: init: 0.030 s, pass 0: 0.030 s, pass 1: 0.000,
04    pass2: 0.000, calc: 0.010 s
05    PROF: job ticket calculation: init: 0.000 s, pass 0: 0.000 s, pass 1: 0.000,
06    pass2: 0.000, calc: 0.000 s
07    PROF: normalizing job tickets took 0.010 s
08    PROF: create active job orders: 0.010 s
09    PROF: job-order calculation took 0.090 s
10    PROF: job sorting took 0.090 s
11    PROF: job dispatching took 0.000 s (20 fast, 0 fast_soft, 0 pe, 0 pe_soft, 0 res)
12    PROF: parallel matching   global   rqs     cqstatic      hstatic
13    qstatic     hdynamic     qdyn
14    PROF: sequential matching global   rqs     cqstatic      hstatic
15    qstatic     hdynamic     qdyn
16    PROF: parallel matching     0       0            0            0
17    0            0            0
18    PROF: sequential matching  20       0           20           20
19    20          20           20
20    PROF: create pending job orders: 0.050 s
21    PROF: scheduled in 0.310 (u 0.220 + s 0.000 = 0.220): 20 sequential, 0 parallel,
22    11799 orders, 3 H, 0 Q, 2 QA, 11775 J(qw), 20 J(r), 0 J(s), 0 J(h), 0 J(e),
23    0 J(x), 11795 J(all), 52 C, 1 ACL, 1 PE, 1 U, 1 D, 1 PRJ, 0 ST, 1 CKPT, 0 RU,
24    1 gMes, 0 jMes, 11799/4 pre-send, 0/0/0 pe-alg
25    PROF: send orders and cleanup took: 0.090 (u 0.080,s 0.000) s
26    PROF: schedd run took: 0.720 s (init: 0.000 s, copy: 0.200 s, run:0.490, free:
27    0.000 s, jobs: 10929, categories: 1/0)
```

The text box above shows the profiling output of one scheduler run.

- Line 1: At the beginning, the scheduler thread receives events containing all information about configuration and state changes since the last event package was received. This line shows how many events the scheduler received and how long it took to update scheduler internal data structures according the instructions in the events.
- Line 2: Shows the time needed to calculate the numbers for the urgency policy.
- Line 3: The output contains different calculation times for the ticket policy. *init* shows how long it took to setup all internal data structures. *Pass 0* to *pass 2* show time for data preparation steps, and *calc* shows the time for the final ticket calculation of all pending jobs.
- Line 4: Same as in line 3 but for running jobs.
- Line 5: Shows the time needed to normalize the tickets so that they are in a range between 0 and 1.
- Line 6: Here, orders for running jobs are generated and sent to other threads executing those orders. The time does not include processing of those orders.
- Line 7: Overall time needed (including all times from 2 to 6) to compute the priority of of all jobs.
- Line 8: Jobs need to be sorted to reflect the job priority. This shows the length of time that this sorting took.
- Line 9: Now the scheduler can start to dispatch jobs to needed compute resources. The time for this step is shown along with how many jobs of each category could be scheduled. The scheduler distinguishes between:
  - *fast* jobs (sequential jobs without soft resource request)
  - *fast_soft* jobs (sequential jobs with soft resource requests)
  - *pe* jobs
  - *pe_soft* jobs (parallel jobs with soft resource requests)
  - *res* jobs (jobs with reservation)
- Line 10-13: Show for how many jobs the different parts of the scheduler algorithm where passed.
- Line 14: Time needed to create priority update orders for all pending jobs.
- Line 15-17: Time (wallclock, system and user time) needed to schedule all jobs including all previous steps except for step 1.
- Line 18: The scheduler already sent orders during the scheduling run. This line shows how long it took to send orders that could not be sent during the scheduler processing, and the time also includes cleanup time to remove data structures that are no longer needed.
- Line 19: The time needed for the whole scheduling run including all previous steps.
  - *init* - initialization time
  - *copy* - time to replicate and filter data for the scheduler processing
  - *run* - scheduler algorithm
  - *free* - time to free previously allocated data
  - *jobs* - number of jobs in the system (before *copy* operation)
  - *categories 1* - number of categories
  - *categories 2* - number of priority classes

The scheduler also dumps system user and wall-clock times of each processing layer.

```
01    PROF(1664087824): scheduler thread profiling summary:
02    PROF(1664087824): other          : wc =  55.870s, utime =   0.000s, stime =
03    6.050s, utilization =  11%
04    PROF(1664087824): packing        : wc =   0.000s, utime =   0.000s, stime =
05    0.000s, utilization =   0%
06    PROF(1664087824): eventclient    : wc =   0.000s, utime =   0.000s, stime =
07    0.000s, utilization =   0%
08    PROF(1664087824): mirror         : wc =   0.020s, utime =   0.000s, stime =
09    0.040s, utilization = 200%
10    PROF(1664087824): gdi            : wc =   0.000s, utime =   0.000s, stime =
11    0.000s, utilization =   0%
12    PROF(1664087824): ht-resize      : wc =   0.000s, utime =   0.000s, stime =
13    0.000s, utilization =   0%
14    PROF(1664087824): scheduler      : wc =   0.910s, utime =   0.340s, stime =
15    0.040s, utilization =  42%
16    PROF(1664087824): pending ticket : wc =   0.130s, utime =   0.000s, stime =
17    0.000s, utilization =   0%
18    PROF(1664087824): job sorting    : wc =   0.110s, utime =   0.120s, stime =
19    0.020s, utilization = 127%
20    PROF(1664087824): job dispatching: wc =   0.000s, utime =   0.000s, stime =
21    0.000s, utilization =   0%
22    PROF(1664087824): send orders    : wc =   0.380s, utime =   0.360s, stime =
23    0.050s, utilization = 108%
24    PROF(1664087824): scheduler event: wc =   0.180s, utime =   0.110s, stime =
25    0.010s, utilization =  67%
26    PROF(1664087824): copy lists     : wc =   1.270s, utime =   0.260s, stime =
27    0.440s, utilization =  55%
28    PROF(1664087824): total          : wc =  60.340s, utime =   2.070s, stime =
29    6.790s, utilization =  15%
```

Activating Scheduler Monitoring

There are different ways to monitor the scheduler and the decisions it makes. Profiling that shows the main activity steps and corresponding run times can be enabled as outlined in the previous chapter. Besides that, administrators can also enable additional monitoring. The monitoring output can be used to find out why certain scheduler decisions where made and also why specific jobs were not started. Note that enabling additional monitoring might throttle down the scheduler and therefore the cluster throughput.

Find Reasons Why Jobs are Not Started

The scheduler can collect the reasons why jobs could not be scheduled during a scheduler run. The parameter `schedd_job_info` of the scheduler configuration enables or disables this functionality. If it is enabled, then messages containing the reasons why it was not possible to schedule a job will be collected for the not-scheduled jobs. The amount of memory that might be needed to store that information within the `sge_qmaster` process could be immense. Due to this reason, this scheduler job information is disabled by default.

If it is enabled, then `qstat` might be used to retrieve that information for a specific jobs:

```
# qstat -j <jid>
```

```
scheduling info: queue instance "all.q@host1" dropped because it is overloaded: ....
                              queue instance "all.q@host1" dropped because it is disabled
                              All queues dropped because of overload or full
                              Job is in hold state
```

Enable Monitoring to Observe Scheduler Decisions

Especially when resource or advance reservation is used in a cluster it might be helpful to understand how the scheduler is influenced by the existing reservations. For this purpose, the scheduler configuration parameter setting `MONITOR` can be enabled. This causes the scheduler to add information to the *schedule* file that is located in the directory `$SGE_ROOT/$SGE_CELL/common/`. The following example briefly introduces scheduler monitoring.

Assume the following sequence of jobs:

```
qsub -N L4_RR -R y -l h_rt=30,license=4 -p 100  $SGE_ROOT/examples/jobs/sleeper.sh 20
qsub -N L5_RR -R y -l h_rt=30,license=5         $SGE_ROOT/examples/jobs/sleeper.sh 20
qsub -N L1_RR -R y -l h_rt=31,license=1         $SGE_ROOT/examples/jobs/sleeper.sh 20
```

These jobs are being submitted into a cluster with the global *license* consumable resource that has been limited to a number of 5 licenses. Due to the use of these default priority settings in the scheduler configuration:

```
weight_priority                  1.000000
weight_urgency                   0.100000
weight_ticket                    0.010000
```

the -p priority of the L4_RR job will be sure to overwhelm the license based urgency, finally resulting in a prioritization such as the following:

```
job-ID  prior   name
---------------------
3127 1.08000 L4_RR
3128 0.10500 L5_RR
3129 0.00500 L1_RR
```

In this case, traces of those jobs can be found in the *schedule* file for 6 schedule intervals:

```
  ::::::::
  3127:1:STARTING:1077903416:30:G:global:license:4.000000
  3127:1:STARTING:1077903416:30:Q:all.q@host3:slots:1.000000
  3128:1:RESERVING:1077903446:30:G:global:license:5.000000
  3128:1:RESERVING:1077903446:30:Q:all.q@host2:slots:1.000000
  3129:1:RESERVING:1077903476:31:G:global:license:1.000000
  3129:1:RESERVING:1077903476:31:Q:all.q@host1:slots:1.000000
  ::::::::
  3127:1:RUNNING:1077903416:30:G:global:license:4.000000
  3127:1:RUNNING:1077903416:30:Q:all.q@host3:slots:1.000000
```

```
3128:1:RESERVING:1077903446:30:G:global:license:5.000000
3128:1:RESERVING:1077903446:30:Q:all.q@host1:slots:1.000000
3129:1:RESERVING:1077903476:31:G:global:license:1.000000
3129:1:RESERVING:1077903476:31:Q:all.q@host1:slots:1.000000
::::::::
3128:1:STARTING:1077903448:30:G:global:license:5.000000
3128:1:STARTING:1077903448:30:Q:all.q@host3:slots:1.000000
3129:1:RESERVING:1077903478:31:G:global:license:1.000000
3129:1:RESERVING:1077903478:31:Q:all.q@host2:slots:1.000000
::::::::
3128:1:RUNNING:1077903448:30:G:global:license:5.000000
3128:1:RUNNING:1077903448:30:Q:all.q@host3:slots:1.000000
3129:1:RESERVING:1077903478:31:G:global:license:1.000000
3129:1:RESERVING:1077903478:31:Q:all.q@host1:slots:1.000000
::::::::
3129:1:STARTING:1077903480:31:G:global:license:1.000000
3129:1:STARTING:1077903480:31:Q:all.q@host3:slots:1.000000
::::::::
3129:1:RUNNING:1077903480:31:G:global:license:1.000000
3129:1:RUNNING:1077903480:31:Q:all.q@host3:slots:1.000000
::::::::
```

For a schedule interval, each section shows all resource utilizations that were taken into account. The *RUNNING* entries show utilizations of jobs that already were running at the beginning of the interval, *STARTING* entries denote immediate utilizations that were decided within the interval, and *RESERVING* entries show utilizations that are planned for the future i.e. reservations.

The format of the schedule file is

- jobid: The jobs id.
- taskid: The array task id or 1 in case of non-array jobs.
- state: One of RUNNING/SUSPENDED/MIGRATING/STARTING/RESERVING.
- start_time: Start time in seconds after 1.1.1970.
- duration: Assumed job duration in seconds.
- level_char: One of *P*,*G*,*H* and *Q* standing for *PE*, *Global* ,*Host* and *Queue*.
- object_name: The name of the PE/global/host/queue.
- resource_name: The name of the consumable resource.
- utilization: The resource utilization debited for the job.

A line "::::::::" marks the begin of a new schedule interval.

Activate Debugging Output from the Command-Line and How to Interpret It

To activate debugging output of Univa Grid Engine applications, do the following before starting the application to be tested:

```
# . $SGE_ROOT/$SGE_CELL/common/settings.sh
# . $SGE_ROOT/util/dl.sh
# dl <debug_level>
# <uge_command>
```

On Windows, it is:

```
> %SGE_ROOT%\%SGE_CELL%\common\settings.bat
> %SGE_ROOT%\util\dl.bat <debug_level>
> <uge_command>
```

The `dl.sh` script makes the `dl` command available. The `dl` command will set necessary environment variables for a specific debug level. If the Univa Grid Engine command is started, then it will print debug messages to stderr. In *debug_level* 1, the applications print general information messages about what steps are executed. *debug_level* 2 will show function calls of the upper processing layers and corresponding locations in the source code that are passed. Other *debug_levels* are available, but are not recommended for users or administrators.

Here is an example for the output of the `qstat` command in *debug_level* 1:

```
01     0  17230 140106943756032  returning port value: 5001
02     1  17230         main     creating qstat GDI handle
03     2  17230         main     file "/Users/ernst/Test/5000/default/
04                                    common/sge_qstat" does not exist
05     3  17230         main     file "/Users/ernst/.sge_qstat" does not exist
06     4  17230         main     queues not needed
07     5  17230         main     sge_set_auth_info: username(uid) = user1(500),
08                                 groupname = univa(1025)
09     6  17230         main     ------- selecting queues -----------
10     7  17230         main     ------- selecting jobs -----------
11     8  17230         main     Destroy handler
```

The first column in the output shows a line number followed by the PID of the process that is being debugged. The third column will either show an internal thread id or the thread name of the thread that logs the message. After that, the debug message is printed.

Activate Debugging Output of Univa Grid Engine Windows services

To activate debugging output of the Univa Grid Engine Windows services `Univa Grid Engine Job Starter Service` and `Univa Grid Engine Starter Service` and the associated `SGE_Starter.exe` binary, follow these steps:

- At the Windows execution host, open the `Services` dialog from the `Control Panel` as an Administrator.
- Open the `Properties` dialog of the Univa Grid Engine Windows service.
- Stop the service.
- Enter "/log" to the `Start parameters:` text field.
- Start the service using the `Start` button on this dialog.

From now on, the services writes a log file to `c:\tmp`. If logging was enabled for the `Univa Grid Engine Job Starter Service`, also the `SGE_Starter.exe` starts writing a log file for each time it is started.

Activate Debugging Output of the `qloadsensor.exe`

If the Windows execution host does not report load, it could be because the `qloadsensor.exe` does not work properly. First check the Windows Task Manager if the qloadsensor.exe runs at all. If it runs and the execution daemon does not report load after more than one minute, test the `qloadsensor.exe` itself. To do this, stop the execution daemon using

```
> qconf -ke <hostname>
```

and check the Windows Task Manager to make sure neither the Windows execution daemon `sge_execd.exe` nor the load sensor `qloadsensor.exe` are running anymore.

To test the `qloadsensor.exe`, open a console window (also called cmd.exe window) as the user that starts the Windows execution daemon. In this console window, run the `$SGE_ROOT/$SGE_CELL/common/settings.bat` file to set the environment variables properly. Then start the load sensor manually:

```
> %SGE_ROOT%\bin\win-x86\qloadsensor.exe
```

Press `Enter` two times and wait some seconds, the load sensor should print an output like this:

```
begin
wega:num_proc:1
wega:load_short:0.010
wega:load_medium:0.010
wega:load_long:0.010
wega:load_avg:0.010
wega:cpu:0
wega:swap_free:1113124864
wega:swap_total:1308422144
wega:swap_used:195297280
wega:mem_free:279228416
wega:mem_total:536330240
wega:mem_used:257101824
wega:virtual_free:1392353280
wega:virtual_total:1844752384
wega:virtual_used:452399104
end
```

If it does not look like this or if the “load_*" values are always 0, enable debugging:

```
> %SGE_ROOT%\bin\win-x86\qloadsensor.exe -set-trace-file c:\tmp\qloadsensor.trace
```

Again, press `Enter` at least two times and wait several seconds. Now the log file `c:\tmp\qloadsensor.trace` should contain the program flow and possibly also error message.

3 Special Activities

3.1 Tuning Univa Grid Engine for High Throughput

In clusters with high throughput, there is usually a high volume of short jobs (job run times in the magnitude of seconds).

Both the submission rate as well as the number of jobs finishing in a certain time frame is high, There may also be a high number of pending jobs.

Cluster sizes range from small clusters with only a few hosts to large clusters with thousands of hosts.

A number of setup and tuning parameters can help in achieving high throughput and high cluster utilization in such high throughput scenarios.

3.1.1 sge_qmaster Tuning

Installation Options

Univa Grid Engine 8.2 allows to activate a read-only-component during installation. If enabled this component will handle all read-only-requests in parallel to read-write-requests. Up to 64 threads can be enabled in this component. Enabling ensures faster response times for all requests which has also a huge positive impact on the cluster throughput.

It is recommended to enable this component during the installation with at least 4 threads. If memory constraints allow to start more threads then this will be helpful especially in huge clusters with several thousand of execution and submit hosts. Find more information concerning read-only-threads and memory requirements in section 'Selecting Thread Setup of the Master Host Component' of the installation guide.

If the read-only-component was disabled during the Univa Grid Engine installation process then it can be manually enabled by adjusting the `reader` parameter located in the `bootstrap` file of the configuration. Adjusting this parameter requires restart of the `sge_qmaster` process.

During the runtime of an instance of `sge_qmaster` with enabled read-only-component it is possible to add/kill reader threads with the `qconf -at reader` and `qconf -kt reader` commands. Increasing the number of reader threads will be helpful for the Univa Grid Engine system when the number of incoming read-only-requests cannot be handled immediately by the active number of threads. In this case waiting requests are added to an `sge_qmaster` internal queue. The length of the queue will be shown in the logging output of qmaster monitoring when this is enabled by setting the `MONITOR_TIME` parameter in the `qmaster_params` of the global configuration.

```
03/20/2014 11:36:40.910387|        listener|v06|P|listener001: runs: 0.25r/s (
    in (g:204.25,a:0.00,e:0.00,r:0.00)/s other (wql:0,rql:10,wrql:0))
    out: 0.00m/s APT: 0.0000s/m idle: 100.00% wait: 0.00% time: 3.97s
```

The `rql` parameter in the `other` section of the monitoring output shows the reader queue length. In this example the queue length is 10 which means that 10 additional threads would be required so that all read-only-requests could be handled immediately.

It is recommended to observe the cluster over longer time to find out what the optimal number of read-only-threads is that should be started. Just increasing the number of read-only-threads to the maximum of 64 is not beneficial because internal locking might slowdown processing in situations of lesser load.

Spooling Options

In high throughput scenarios, performance of the cluster highly depends on the spooling done by sge_qmaster. Every job submission, job status transition and finally job end result in spooling operations.

Therefore the sge_qmaster spooling options should be carefully chosen:

- Use Berkeley DB spooling if possible.
- Do spooling on a local file system, unless a high availability is required using sge_shadow, see also Ensuring High Availability.
- If spooling needs to be on a shared file system, Berkeley DB spooling on NFS4 is preferred over classic spooling.

Choosing the spooling method is usually done during Univa Grid Engine installation.

Configuration Options

The following options in the global cluster configuration can have a significant impact on sge_qmaster performance. Changing these parameters takes immediate effect.

- The attribute *loglevel* defines how much information is logged to the sge_qmaster messages file during sge_qmaster runtime. If *loglevel* is set to *log_info*, messages will get logged at every job submission and job termination. Set *loglevel* to *log_warning* to reduce overhead from writing the sge_qmaster messages file.
- Do the following configuration for the attribute *reporting_params*:
  - Make sure to write operations on the accounting file and optionally if the reporting files are buffered. The parameter *flush_time* should be set to at least one second (00:00:01). If it is set to 0, buffering of write operations to the accounting and the reporting file is not done. Should the attribute *accounting_flush_time* be set, it must either be removed (meaning that *flush_time* will be in effect for the accounting file) or set to at least one second (00:00:01).
  - If reporting is enabled, the *log_consumables* attribute must be set to *false*. *log_consumable* is an option ensuring compatibility to a (mis)behavior in Sun Grid Engine < 6.2. Setting it to *true* results in a high volume of data written to the reporting file whenever a consumable value changes. It should always be set to *false*.

See also Understanding and Modifying the Cluster Configuration for more details on the global cluster configuration.

3.1.2 Tuning Scheduler Performance

An important factor in high throughput scenarios is scheduler performance. Reducing the time required for a single scheduling run will allow for more precise scheduling runs.

The scheduler configuration allows for the setting of attributes having significant impact on scheduler performance.

- Setting the attribute *flush_submit_sec* to 1 triggers a scheduling run whenever a job is submitted. Given that there are free resources in the cluster the newly submitted job might get started immediately.
- The attribute *flush_finish_sec* has a similar meaning. By settings its value to 1 a scheduling run is triggered whenever a job finishes. The resources having been held by the just finished job can get reused immediately.
- The default configuration of Univa Grid Engine makes scheduler dispatch jobs to the least loaded host and adds some virtual load to a host when a job gets dispatched to it. Adding virtual load to a host requires sorting the host list after every dispatch operation, which can be an expensive operation in large clusters. By setting the attribute *load_adjustment* to *NONE* scheduling overhead can be reduced significantly.
- When the *schedd_job_info* attribute is set to *true* scheduler provides information about why a job cannot get dispatched to sge_qmaster which can then be queried by calling `qstat -j <job_id>`. Setting *schedd_job_info* to *false* significantly reduces the amount of information generated by scheduler and being held by sge_qmaster, lowering the amount of memory required by sge_qmaster and the overhead of producing the information messages. Querying the reason why a job cannot get dispatched is then provided by calling `qalter -w p <job_id>`.
- Resource reservation results in quite expensive analysis being done in the scheduler. If resource reservation is not required, consider disabling it completely by setting the attribute *max_reservation* to 0.

See also Understanding and Modifying the Univa Grid Engine Scheduler Configuration for further information about the scheduler configuration.

In general, the fewer the scheduling policies configured, the higher the scheduler performance.

3.1.3 Reducing Overhead on the Execution Side

Local sge_execd Spooling

In high throughput scenarios with short running jobs, many jobs are started and completed per time period. One of the most expensive operations at job start and end is job spooling and the creation of temporary files and directories for the job start and the cleaning of temporary data at job end.

By configuring the execution daemons to use a local file system for spooling, performance can be significantly improved.

For changing the sge_execd spool directory

- make sure no jobs are running on the hosts affected,
- modify the global cluster configuration or the local cluster configuration for the exec host,
- set the attribute execd_spool_dir to the new spool directory,
- shut down and restart the sge_execd.

Setting Execution hosts to use Extended Memory Data Metrics

The default memory collection metrics in Univa Grid Engine can be extended on modern Linux operating systems to include additional memory metrics added to Kernel 2.6.25 or later. The `ENABLE_MEM_DETAILS` execd parameter flag can be set in the global cluster configuration using `qconf -sconf`. When `ENABLE_MEM_DETAILS` is set to 1 or `TRUE` Univa Grid Engine will collect additional per job memory usage for resident set size (rss), proportional set size (pss), shared memory (smem) and private memory (pmem).

3.2 Optimizing Utilization

Cluster utilization describes the proportion of resources currently used by Univa Grid Engine jobs in comparison to the whole amount of available resources installed at the compute cluster. Reaching high cluster utilization is one of the main goals which Univa Grid Engine has on its agenda. This section describes basic techniques for optimizing the resource utilization of a Univa Grid Engine managed cluster.

3.2.1 Using Load Reporting to Determine Bottlenecks and Free Capacity

In order to provide a quick overview about the clusters compute resources state, the `qhost` command can be used. Interesting values are the current load value (**LOAD**) and the amount of memory currently in use (**MEMUSE**).

```
> qhost
HOSTNAME  ARCH    NCPU         NSOC NCOR NTHR  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------------
global              -           -    -    -     -   -     -          -       -       -
host1               lx-amd64    1    1    1    1  0.21  934.9M  147.7M 1004.0M     0.0
host2               lx-x86      1    0    0    0  0.09 1011.3M  103.4M    1.9G     0.0
host3               lx-amd64    2    1    2    2  0.46    3.4G  343.3M    2.0G     0.0
host4               sol-amd64   2    1    2    2  2.07    2.0G  763.0M  511.0M     0.0
host5               lx-amd64    1    1    1    1  0.09  492.7M   75.3M  398.0M
```

Unused hosts can be identified through a low load value. To sort the output by load, use standard commands like the following:

```
> qhost | tail +4 |  sort -k 7
host2            lx-x86          1    0    0    0  0.13 1011.3M  103.4M    1.9G     0.0
host5            lx-amd64        1    1    1    1  0.14  492.7M   75.3M  398.0M     0.0
host1            lx-amd64        1    1    1    1  0.29  934.9M  147.7M 1004.0M     0.0
host3            lx-amd64        2    1    2    2  0.64    3.4G  343.3M    2.0G     0.0
host4            sol-amd64       2    1    2    2  1.94    2.0G  763.0M  511.0M     0
```

More detailed load information can be seen on execution host level. The `qconf -se <hostname>` displays the current **raw** load values.

```
> qconf -se host3
...
load_values           load_avg=0.000000,load_short=0.000000, \
                      load_medium=0.000000,load_long=0.000000,arch=lx-amd64, \
                      num_proc=1,mem_free=2818.867188M,swap_free=2053.996094M, \
                      virtual_free=4872.863281M,mem_total=3144.273438M, \
                      swap_total=2053.996094M,virtual_total=519\UGEShortVersion{}69531M, \
                      mem_used=325.406250M,swap_used=0.000000M, \
                      virtual_used=325.406250M,cpu=0.200000,m_topology=SC, \
                      m_topology_inuse=SC,m_socket=1,m_core=1,m_thread=1, \
                      np_load_avg=0.000000,np_load_short=0.000000, \
                      np_load_medium=0.000000,np_load_long=0.000000
...
report_variables      NONE
```

In order to see the processed (in case of load scaling) values `-h hostname -F` can be used:

```
> qhost -h host6 -F
HOSTNAME         ARCH         NCPU NSOC NCOR NTHR  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------------
global           -             -    -    -    -     -       -       -       -       -
host7          lx-x86          1    0    0    0  0.00 1011.3M  106.5M    1.9G     0.0
  hl:arch=lx-x86
  hl:num_proc=1.000000
  hl:mem_total=1011.332M
  hl:swap_total=1.937G
  hl:virtual_total=2.925G
  hl:load_avg=0.000000
  hl:load_short=0.000000
  hl:load_medium=0.000000
  hl:load_long=0.000000
  hl:mem_free=904.812M
  hl:swap_free=1.937G
  hl:virtual_free=2.821G
  hl:mem_used=106.520M
  hl:swap_used=0.000
  hl:virtual_used=106.520M
  hl:cpu=0.000000
  hl:m_topology=NONE
  hl:m_topology_inuse=NONE
  hl:m_socket=0.000000
  hl:m_core=0.000000
  hl:m_thread=0.000000
  hl:np_load_avg=0.000000
  hl:np_load_short=0.000000
  hl:np_load_medium=0.000000
```

```
   hl:np_load_long=0.000000
```

The current cluster utilization must always be seen in conjunction with the pending job list. If there are no jobs waiting for resources, the utilization is already perfect from the DRM point of view. The `qstat` command gives an overview of running and pending jobs. Running jobs are in state **r** and pending jobs are in state **qw** (for queued waiting). The time of submission is visible, depending on job status and the requested number of slots.

```
> qstat
job-ID  prior   name   user      state  submit/start at     queue  slots ja-task-ID
-----------------------------------------------------------------------------------
  6 0.55500 sleep    daniel       r     04/14/2011 09:45:12 bq@macsuse      1
  7 0.55500 sleep    daniel       r     04/14/2011 09:45:12 bq@macsuse      1
  8 0.55500 sleep    daniel       qw    04/14/2011 09:44:25                 1
  9 0.55500 sleep    daniel       qw    04/14/2011 09:44:25                 1
```

More information about why certain jobs are not scheduler can be also retrieved by the `qstat` command. A prerequisite for this is that in the scheduler configuration the **schedd_job_info** parameter is set to **true**.

Note

Note that enabling the scheduler output has implications to the overall performance of the qmaster process and should be activated either in smaller clusters, where the qmaster host is just slightly loaded or just temporary.

```
> qconf -msconf
...
schedd_job_info                   true
```

When there are any pending jobs the scheduling information can be viewed by a simple `qstat -j <jobno>`

```
 > qstat -j <jobno>
 ...
 scheduling info:     queue instance "all.q@SLES11SP1" dropped because it is full
                      queue instance "all.q@u1010" dropped because it is full
                      queue instance "all.q@cent48" dropped because it is full
                      queue instance "all.q@macsuse" dropped because it is full
                      queue instance "all.q@solaris10" dropped because it is full
```

In the output above, all queue-instances are already full and there are no more slots left.

3.2.2 Scaling the Reported Load

Sometimes load values have different meanings. The machine load average could be such an example. It is defined by the number of processes in the operating systems running queue. On a

multi-cpu or multi-core host, usually multiple processes can be run at the same time thus a load of 1.0 means that it is fully occupied on a one core machine while there are still resources left on a multi-core machine. In order to resolve these issues, load report values can be scaled on host level.

Example: Downscale load_short by a Factor of 10

Load scaling is host-specific therefore the host configuration must be adapted:

```
> qconf -me <hostname>
hostname              <hostname>
load_scaling          load_short=0.10000
...
```

The original execution host source values can still be seen in the host configuration (**load_short=0.08**):

```
> qconf -se <hostname>
hostname              <hostname>
load_scaling          load_short=0.100000
complex_values        NONE
load_values           load_avg=0.060000,load_short=0.080000, \
                      load_medium=0.060000,load_long=0.110000,arch=lx-amd64, \
                      num_proc=1,mem_free=2742.671875M,swap_free=2053.996094M, \
                      virtual_free=4796.667969M,mem_total=3144.273438M, \
                      swap_total=2053.996094M,virtual_total=519\UGEShortVersion{}69531M, \
                      mem_used=401.601562M,swap_used=0.000000M, \
                      virtual_used=401.601562M,cpu=73.800000,m_topology=SC, \
                      m_topology_inuse=SC,m_socket=1,m_core=1,m_thread=1, \
                      np_load_avg=0.060000,np_load_short=0.080000, \
                      np_load_medium=0.060000,np_load_long=0.110000
```

...

The current scaled load values (**load_short=0.008** in comparison to the source **0.08**) are shown by the `qstat`:

```
> qstat -l h=<hostname> -F load\_short
queuename                      qtype resv/used/tot. load_avg   arch       states
---------------------------------------------------------------------------------
all.q@<hostname>                BIPC    0/0/20         0.06  lx-amd64
   hl:load_short=0.008000
```

Note

The scaled load values are already available with the np_load_* values. They are scaled using the number of reported processors (num_proc).

3.2.3 Alternative Means to Determine the Scheduling Order

After a default installation, the scheduler is configured in a way to choose the more available hosts first for the new jobs. The scheduler configuration can be viewed with the `qconf -ssconf` command.

```
> qconf -ssconf
...
job_load_adjustments              np_load_avg=0.50
load_adjustment_decay_time        0:7:30
host_sort_formula                 np_load_avg
schedd_job_info                   false
weight_host_affinity              0.0
weight_host_sort                  1.0
weight_queue_affinity             0.0
weight_queue_host_sort            1.0
weight_queue_seqno                0.0
```

The **weight_queue_*** and **weight_host_***-parameters determine the order of the queue-instances when they are matched against the pending job list. The **host_sort_formula** describes load type and is calculated if **weight_host_sort** is set to something bigger than 0.

Queue Sequence Number

When **weight_queue_seqno** is set to a much higher value than **weight_queue_host_sort**, the queue sequence number, which is defined in the queue configuration attribute (`qconf -mq <queue_name>`) determines the order in which the queued instances are chosen for the pending jobs.

Example: Defining the Queue Order

Create two queues `a` and `b`.

```
> qconf -aq a
> qconf -aq b
```

Disable queue `all.q` if it exists.

```
> qmod -d all.q
```

Set `weight_queue_seqno` to 1 and all other `weight_`-parameters to 0.

```
> qconf -msconf
...
```

weight_host_affinity 0.0 weight_host_sort 0.0 weight_queue_affinity 0.0 weight_queue_host_sort 0.0 weight_queue_seqno 1.0 ...

Set the `seq_no` of queue `a` to 10 and `seq_no` of queue `b` to 20.

```
> qconf -mq a
qname                 a
hostlist              @allhosts
seq_no                10
...

> qconf -mq b
qname                 b
hostlist              @allhosts
seq_no                20
...
```

Submit some jobs. It can be observed that all jobs are running in queue `a`.

```
> qsub -b y sleep 120
> qsub -b y sleep 120
> qsub -b y sleep 120
> qsub -b y sleep 120

> qstat -g c
CLUSTER QUEUE    CQLOAD   USED    RES  AVAIL  TOTAL aoACDS  cdsuE
--------------------------------------------------------------------
a                0.01        4      0     46     50      0      0
all.q            0.01        0      0      0     60      0     60
b                0.01        0      0      5      5      0      0
```

Example: Defining the Order on Queue Instance Level

Defining the queue order on queue level can be too vague when implementing specific scheduling strategies. A queue could span a large number of hosts or even the whole cluster. Therefore it is useful to define sequence numbers on queue instance levels (a queue instance the part of a queue which sits on a specific host).

The order can be defined on queue instance level in the following way:

```
> qconf -mq a
qname                 a
hostlist              @allhosts
seq_no                10,[host1=1],[host2=2],[host3=3]
...
slots                 1
...
```

If 4 jobs are submitted, the first one is dispatched to `host1`, the second to `host2` and so on.

```
> qsub -b y sleep 120
> qsub -b y sleep 120
> qsub -b y sleep 120
> qsub -b y sleep 120
```

```
> qstat
job-ID  prior   name   user    state submit/start at     queue    slots ja-task-ID
-----------------------------------------------------------------------------------
     4 0.55500 sleep  daniel    r     04/28/2011 14:31:36 a@host1         1
     5 0.55500 sleep  daniel    r     04/28/2011 14:31:36 a@host2         1
     6 0.55500 sleep  daniel    r     04/28/2011 14:31:36 a@host3         1
     7 0.55500 sleep  daniel    r     04/28/2011 14:31:36 a@host4         1
```

Here `host1` takes precedence over `host2` in queue `a`, and so on.

Example: Antipodal Sequence Numbering of Queues

A Univa Grid Engine-managed cluster is often populated by jobs with different priorities. In many cases there are several extended I/O intensive (with a low load) batch jobs which are not time sensitive, running simultaneously with a group of high priority jobs which require immediate execution thus halting already-running jobs. In order to configure the cluster for these two job types, two queues have to be added to the configuration. For simplicity, 3 hosts are used within this example.

```
> qconf -mq low
qname                 low
hostlist              @allhosts
seq_no                10,[host1=3],[host2=2],[host3=1]
...
slots                 1
...

> qconf -mq high
qname                 high
hostlist              @allhosts
seq_no                10,[host1=1],[host2=2],[host3=3]
...
slots                 1
...
```

This example shows that the high queue suspends the low queue. The `seq_no` in both configurations is now defined on **queue instance layer** with a reverse order respectively. The net result is that jobs which are submitted to the `high` queue run first `host1` then `host2` and so on and jobs which are running in the `low` queue begin from the opposite end. This means that jobs are suspended only when the cluster is fully utilized. A drawback in this example is the problem of starvation. Low priority jobs which are running on hosts with a very low sequence number for the high priority queue instance can remain suspended for a long time when there are always some high prior jobs running. A more advanced approach is showed in section Implementing Pre-emption Logic with the example **Mixing exclusive high priority jobs with low priority jobs**.

This example shows that the high queue suspends the low queue. The `seq_no` in both configurations is now defined on **queue instance layer** with a reverse order respectively. The net result is that jobs which are submitted to the `high` queue run first `host1` then `host2` and so on and

jobs which are running in the `low` queue begin from the opposite end. This means that jobs are suspended only when the cluster is fully utilized. A drawback in this example is the problem of starvation. Low priority jobs which are running on hosts with a very low sequence number for the high priority queue instance can remain suspended for a long time when there are always some high prior jobs running. A more advanced approach is showed in section Implementing Pre-emption Logic with the example **Mixing exclusive high priority jobs with low priority jobs**.

3.3 Managing Capacities

Administrators are often faced with the problem that the number of resources used at the same point in time has to be limited for different consumers in order to map given business rules into a Univa Grid Engine cluster installation. Univa Grid Engine includes several modules for limiting capacities of the managed resources. The main concepts to ensure these limits in Univa Grid Engine are the **resource quota sets** and the **consumables**, which are illustrated in more detail below.

3.3.1 Using *Resource Quota Sets*

With resource quota sets the administrator is able to restrict different objects, like users, projects, parallel environments, queues, and hosts with a different kind of limit. Limits can be static, with a fixed value, or dynamic, which is a simple algebraic expression. All currently-defined resource quota sets can be shown with the `qconf -srqsl`. After a default installation, no resource quota set is defined.

```
>qconf -srqsl
no resource quota set list defined
```

Resource quotas can be added with`qconf -arqs`, modified with `qconf -mrqs myname`, and deleted with `qconf -drqs myname`.

A resource quota set has the following basic structure:

```
{
   name           myresourcequotaset
   description    Just for testing.
   enabled        TRUE
   limit          users {*} to slots=2
}
```

The **name** denotes the name of the rule, it should be short and informative because this name can be seen in the `qstat -j <jobno>` output as the reason why a specific job is not processed in the last scheduled run (Note: `schedd_job_info` must be turned on in the scheduler (see TODO)).

```
>qstat -j 3
...
scheduling info:            cannot run because it exceeds limit "/////"
                            in rule "myresourcequotaset/1"
...
```

The **description** can contain more detailed information about the limits. This becomes important especially when the cluster configuration grows in order to keep track of all defined rules. It should describe the rules in a way that even after years the purpose of the rules can be seen immediately.

The **enabled** field determines whether the rule is enabled (`TRUE`) or disabled (`FALSE`). Hence rules do not have to be deleted and restored as a whole, but can turned off and on, simplifying the handling of resource quota sets.

The entry which defines a rule starts with the keyword **limit**. In the first example above, each user is limited to the use of 2 slots at a time. If a user has, for example, 3 jobs submitted, one job will stay in waiting state (`qw` state) until the first job finishes. The scheduler ensures that not more than 2 slots are occupied by one user at the same time.

It is allowed to arrange more limit rules among one another. If multiple limits match, the first match wins.

```
{
  name          myresourcequotaset
  description   Just for testing
  enabled       TRUE
  limit         users {*} to slots=2
  limit         users {*} to slots=1
}
```

The first limit in this example which matches a user is `users to slots=2` hence a user is allowed to run 2 sequential jobs in parallel.

Limits can have a name which must be unique within one resource quota set:

```
{
  name          myresourcequotaset
  description   Just for testing
  enabled       TRUE
  limit         name slot2rule users {*} to slots=2
  limit         name slot1rule users {*} to slots=1
}
```

Objects, which can be restricted are **users**, **projects**, **pes**, **queues**, and as well as **hosts**. In order to specify entities within these objects, the `{ }` notation can be used. Special values are the asterisk *{}, which means **all**, the exclamation mark {!}, which can be used to exclude entities, as well as the combination of both {!}*, which are all entities which have not requested the specific object.

In the following example user1 and user2 (each of them) are restricted to use 2 slots of the parallel environment mytestpe at the same time.

```
{
  name          myresourcequotaset
  description   Just for testing
```

```
  enabled       TRUE
  limit         users {user1,user2} pes { mytestpe } to slots=2
}
```

In order to limit all users to have at most 100 serial job running in the system, but unlimited parallel jobs, the rule below can be used.

```
{
  name          myresourcequotaset
  description   Just for testing
  enabled       TRUE
  limit         users {*} pes {!*} to slots=100
}
```

The limit after the **to** keyword can be any complex (see TODO) defined in the system. In order to define rules, which are different for specific hosts, dynamic complexes can be used.

The following example limits the number of slots on each host to the number of available cores:

```
{
  name          myresourcequotaset
  description   Just for testing
  enabled       TRUE
  limit         hosts {*} to slots=$m_core
}
```

3.3.2 Using *Consumables*

Consumables are complexes with an additional counting behavior. They can be identified through the consumable column, when displaying the complexes with `qconf -sc`. In a default installation only one (special) consumable is defined - the `slots` complex.

```
>qconf -sc
#name  shortcut   type        relop requestable consumable default  urgency
#--------------------------------------------------------------------------
...
slots     s       INT          <=      YES         YES        1      1000
...
```

The best way to think about consumables is to consider them as counter variables, which can have any semantic one can imagine. These consumables can be defined on different layers: When they are initialized on the **host level** they can limit the number of consumers on specific hosts. If they are initialized in **queues** they limit the use of specific queue instances, and when they are added in **global** configuration (`qconf -me global`) they limit the usage of this resource for each job.

A common task is to handle special hardware devices for a cluster and to make them available for the Univa Grid Engine. In the following example, a group of execution hosts are upgraded with GPU cards in order to support special numerical computational jobs.

Host Consumable Example: Adding a GPU into the cluster

In the current cluster, 3 execution hosts are defined and one of them `host1` with the additional GPU processing facility.

```
> qhost
HOSTNAME                ARCH         NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------
global                  -               -     -       -       -       -       -
host1                   lx26-amd64      4  0.00  934.9M  134.1M 1004.0M     0.0
host2                   lx26-amd64      4  0.02    2.0G  430.9M    2.0G     0.0
host3                   lx26-amd64      4  0.00  492.7M   41.6M  398.0M     0.0
```

As the first step a new consumable must be added in the complex table.

```
> qconf -mc
#name      shortcut   type        relop requestable consumable default  urgency
#----------------------------------------------------------------------------------
GPU        gpu        INT         <=    YES         YES        0        1000
```

Because this consumable is host-dependent (and not queue-dependent), it must be initialized per host. The execution server configuration must be edited and the new GPU complex value 1 is added.

```
> qconf -me host1
hostname              host1
load_scaling          NONE
complex_values        GPU=1
user_lists            NONE
xuser_lists           NONE
projects              NONE
xprojects             NONE
usage_scaling         NONE
report_variables      NONE
```

Now the value can be seen by the `qstat` output:

```
> qstat -F GPU

queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@host1                    BIPC  0/0/10         0.00     lx26-amd64
      hc:GPU=1
---------------------------------------------------------------------------------
all.q@host2                    BIPC  0/0/10         0.00     lx26-amd64
---------------------------------------------------------------------------------
all.q@host3                    BIPC  0/0/10         0.00     lx26-amd64
```

In order to request the GPU consumable the user must specify the attribute on job submission time.

```
> qsub -b y -l GPU=1 sleep 100
Your job 4 ("sleep") has been submitted
```

Now check the host consumable again:

```
> qstat -F GPU
queuename                qtype resv/used/tot. load_avg arch        states
---------------------------------------------------------------------------
all.q@host1              BIPC    0/1/10         0.00  lx26-amd64
       hc:GPU=0

4 0.55500 sleep      daniel       r     03/04/2011 10:17:21     1
---------------------------------------------------------------------------------
all.q@host2                    BIPC  0/0/10         0.00     lx26-amd64
---------------------------------------------------------------------------------
all.q@host3                    BIPC  0/0/10         0.00     lx26-amd64
```

If a second GPU job is started, then let the scheduler run again (-tsm):

```
> qsub -b y -l GPU=1 sleep 100
Your job 5 ("sleep") has been submitted

> qconf -tsm
daniel@hostname triggers scheduler monitoring

> qstat
job-ID  prior   name  user       state   submit/start at     queue   slots ja-task-ID
-----------------------------------------------------------------------------------
     4 0.55500 sleep  daniel       r     03/04/2011 10:17:21 all.q@host1       1
     5 0.55500 sleep  daniel       qw    03/04/2011 10:17:28
```

The second job, which requests a GPU stays in waiting state until the first GPU job finishes, since there is no other host with a GPU consumable configured.

```
> qstat
job-ID  prior   name   user    state    submit/start at     queue    slots ja-task-ID
-----------------------------------------------------------------------------------
     5 0.55500 sleep  daniel      r     03/04/2011 10:19:07 all.q@host1        1
```

Queue Consumable Example: Adding a multiple GPUs on cluster hosts

This example illustrates how to use queue consumables. Queue consumables can be used when resources should be split up between several queues. Imagine that two GPU cards are added to an execution host. Using the approach above with an higher counter (two instead of one) works

just fine, but the jobs have to negotiate the GPU used (GPU0 or GPU1). One approach to solve this issue would be for the administrator to provide a script on the execution host which then provides the GPU number for the job. In order to handle this with Univa Grid Engine, **queue consumables** can be used.

Like stated above, the GPU complex must first be added in the complex list with `qconf -mc`. In contrast to a host complex, the initial value has to be defined on queue layer. Therefore two queues, each representing one GPU, must be added and initialized properly.

```
> qconf -aq gpu0.q
qname                 gpu0.q
hostlist              host1
...
slots                 10
...
complex_values        GPU=1
...

> qconf -aq gpu1.q
qname                 gpu0.q
hostlist              host1
...
slots                 10
...
complex_values        GPU=1
...
```

The **complex_values** entry can also set different values for each queue instance. If on some hosts GPU pairs should be requestable by just one job the **complex_values** entry could look like the following: `GPU=1,[host2=GPU=2]`.

The **hosts** entry contains all hosts with two GPUs installed, the **complex_values** entry is used for initializing the GPU value. The values can now be seen in the `qstat` output:

```
> qstat -F GPU
queuename                      qtype resv/used/tot. load_avg arch  states
---------------------------------------------------------------------------
all.q@host1                    BIPC  0/0/20         0.23     lx-amd64
---------------------------------------------------------------------------
all.q@host2                    BIPC  0/0/10         0.08     lx-x86
---------------------------------------------------------------------------
all.q@host3                    BIPC  0/0/10         0.04     lx-amd64
---------------------------------------------------------------------------
all.q@host4                    BIPC  0/0/10         0.04     lx-amd64
---------------------------------------------------------------------------
gpu0.q@host1                   BIP   0/0/10         0.23     lx-amd64
       qc:GPU=1
---------------------------------------------------------------------------
gpu1.q@host1                    BIP   0/0/10         0.23     lx-amd64
       qc:GPU=1
```

Now jobs can be submitted with requesting the queue consumable:

```
> qsub -S /bin/bash -l gpu=1 gpu.sh
```

The `gpu.sh` is like the following:

```
#!/bin/bash

if [ "x$QUEUE" = "xgpu0.q" ]; then
    echo "Using GPU 0"
fi

if [ "x$QUEUE" = "xgpu1.q" ]; then
    echo "Using GPU 1"
fi

sleep 100
```

After the job is scheduled the `qstat` shows, which queue and therefore which GPU is selected:

```
> qstat -F gpu
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@host1                    BIPC  0/0/20         0.23     lx-amd64
---------------------------------------------------------------------------------
all.q@host2                    BIPC  0/0/10         0.08     lx-x86
---------------------------------------------------------------------------------
all.q@host3                    BIPC  0/0/10         0.04     lx-amd64
---------------------------------------------------------------------------------
all.q@host4                    BIPC  0/0/10         0.04     lx-amd64
---------------------------------------------------------------------------------
gpu0.q@host1                   BIP   0/1/10         0.05     lx-amd64
       qc:GPU=0
      5 0.55500 gpu.sh     daniel       r     04/19/2011 08:58:08     1
---------------------------------------------------------------------------------
gpu1.q@host1                    BIP   0/0/10         0.05     lx-amd64
       qc:GPU=1
```

The output of the job is:

```
Using GPU 0
```

When 3 jobs are submitted each requesting a GPU, only 2 will run at the same time. The third one is rejected because 2 GPUs are available. If the scheduler information is turned on (`qconf -msconf`) the reason why the third job remains pending can be seen immediately:

```
> qstat -j <jobno>
...
       (-l GPU=1) cannot run in queue "gpu0.q@host1" because it offers only qc:GPU=0.000000
       (-l GPU=1) cannot run in queue "gpu1.q@host1" because it offers only qc:GPU=0.000000
```

3.4 Implementing Pre-emption Logic

Pre-emption is the action of suspending a job in order to free computational resources and resume the job at a later time. The reasons can be different: to avoid thrashing or to give jobs of higher priority precedence. Pre-emption can be configured in Univa Grid Engine on different places and can have different meanings. Limits can be set between different queues so that once queue gains precedence over another, jobs within the queue of higher priority can trigger the suspension of jobs of lower priority in the queue. Furthermore, suspension thresholds within a queue can be defined with the result that whenever the limits are exceeded, jobs are suspended. Additionally the Univa Grid Engine calendar is able to suspend resources. In this section the queue-wise suspension feature is explained in detail.

3.4.1 When to Use Pre-emption

Queue wise subordination can be used whenever jobs have to be grouped into different priority classes. Jobs of a certain class are submitted into the corresponding queue. Whenever a certain limit of jobs of high priority in a queue has been reached, the jobs of lower priority (i.e. the jobs within the subordinate queues) are suspended. After the jobs of higher priority are completed, the suspended jobs will be reinstated. Suspending and reinstating jobs is usually performed by sending the `SIGSTOP` and `SIGCONT` signal to the user jobs. In the queue configuration attribute `suspend_method` and `resume_method` the path to a self-defined script/executable can be added, which overrides the default signals with a user defined suspension/reinstatement behavior. Within this script, different suspend/resume methods for different jobs can be defined. In the case that a different signal for all jobs is needed, the job signal name (SIG*) can be used.

3.4.2 Utilizing Queue Subordination

Queue subordination is defined in the queue configuration, which can be modified through the `qconf -mq <queuename>` command. The related attribute for subordination definitions is named `subordinate_list`. The syntax is:

```
<queuename>=<slots>, <queuename2>=<slots>, ...
```

where the queue name denotes the subordinated queue (the lower priority queue) and slots is the threshold value which triggers the suspension of the subordinate queue.

In case the limits should be different for specific hosts, the following syntax can be used:

```
<queuename>=<default_slots>, [<queuename>@<host>=<slots>]
```

Example: Suspend all low priority jobs on a host whenever a job is running in the high priority queue

First create the low priority queue:

```
> qconf -aq low.q
```

```
qname               low.q
hostlist            @allhosts
slots               10
```

Create the high priority queue with a slot limit of 1 (when 1 slot is used in the upper queue to suspend the lower queue).

```
> qconf -aq high.q

qname               high.q
hostlist            @allhosts
slots               10
subordinate_list    low.q=1
```

Now submit a job into the subordinate queue on `host1`:

```
> qsub -q low.q@host1 -b y sleep 240
```

See that the job is running:

```
> qstat
job-ID  prior   name       user    state  submit/start at     queue  slots ja-task-ID
-----------------------------------------------------------------------------------
     4 0.55500 sleep      daniel    r     05/17/2011 15:36:04 low.q@host1     1
```

Submit the high priority job:

```
> qsub -q high.q@host1 -b y sleep 240
```

After the job is dispatched, the job in the lower priority queue suspends immediately.

```
> qstat
job-ID  prior   name       user      state   submit/start at     queue  slots ja-task-ID
-----------------------------------------------------------------------------------
     4 0.55500 sleep      daniel       S     05/17/2011 15:36:04 low.q@host1      1
     5 0.55500 sleep      daniel       r     05/17/2011 15:36:14 high.q@host1     1
```

3.4.3 Advanced Pre-emption Scenarios

Job suspension can come with the price of a lower overall cluster utilization. The following scenario makes this clear:

The cluster consists of two hosts on which a `high.q` and a `low.q` is defined. The `high.q` subordinates the `low.q` with a limit of 2 which means that whenever two or more jobs are running in the `high.q` on a specific host the `low.q` on that host is subordinated. On both hosts one job is running in the `low.q`. Additionally on `host1`, one job is in `high.q`. If now a second job of higher

priority is submitted, it does not in all cases run on `host2`. If for example the queue sort method is `load` and the two jobs on `host1` produce less load than the one job on `host2`, then the fourth job is scheduled on `host1` with the net result that the job of lower priority is suspended. No suspension would result if the job runs on `host2`.

Usually having the queue instances sorted by load is good way to prevent subordination. But this is not true in all cases. The following example shows how to combine the queue sort method `seq_no` and the exclusive queue feature together with queue-wise subordination.

Example: Mixing exclusive high priority jobs with low priority jobs

In the following scenario, a computer cluster of 8 hosts is used by two different groups: researchers and students. Usually the researchers have just one or two jobs running while the students must do their assignments on the cluster. Therefore two hosts are reserved for the researchers (as students should not to have access to these machines) and the remaining 6 hosts are used by the students. The researchers want to have their machines exclusively if a job (with using 1 or more slots) is running which means a mix of different researcher jobs on one machine is also not allowed. In some rare cases the researchers have much more work, therefore it should be possible that in such circumstances, they can access the machines of the students. But when there are just a few student jobs running, the risk of suspending these jobs should be minimized. All this can be expressed in Univa Grid Engine in the following way:

- because research job need machines exclusively: the exclusive queue complex (consumable) is needed
- two queues are needed: `research.q` and `student.q`
- research jobs should be able to suspend student jobs: queue wise subordination must be configured
- research jobs should first use their own hosts and if this is not enough, student hosts are accessed: queue sort method seq_no
- 3 specific student hosts should be the last resort for research jobs: queue sorting of student queue should be diverted to the sort method of the research hosts

The configuration is done in the following way:

Create `qexclusive` queue consumable (complex).

```
> qconf -mc
#name               shortcut   type        relop requestable consumable default  urgency
#--------------------------------------------------------------------------------------------
qexclusive          qe         BOOL        EXCL  YES         YES        0        4000
```

Create the `student.q`:

```
> qconf -aq student.q
qname                 student.q
hostlist              host3 host4 host5 host6 host7 host8
seq_no                10,[host3=4],[host4=3],[host5=2],[host6=1],[host7=1],[host8=1]
...
slots                 4
```

Create the `research.q` which subordinates `student.q` and define the queue instance exclusive resource:

```
> qconf -aq research.q
qname                 research.q
hostlist              host1 host2 host3 host4 host5
seq_no                10,[host1=1],[host2=1],[host3=2],[host4=3],[host5=4]
slots                 4
...
subordinate_list      student.q=1
complex_values        qexclusive=1
...
```

Change the scheduler configuration:

```
> qconf -ms
...
weight_host_affinity              0.0
weight_host_sort                  0.0
weight_queue_affinity             0.0
weight_queue_host_sort            0.0
weight_queue_seqno                1.0
...
```

Now the researchers have to request the `research.q` together with the `qexclusive` complex and the students have to request `student.q`. This can be enforced by using request files or the `JSV` facility.

3.5 Integrating Univa Grid Engine With a License Management System

Applications run under Univa Grid Engine control may be licensed. In many cases a license management system controls the number of concurrent uses of the application. These licenses are configured as consumable resources in Univa Grid Engine. See also: Introduction Guide -> Concepts and Components -> Expressing Capabilities and Capacities and Special Activities -> Using Consumables.

Once a consumable resource has been created as a license counter, its capacity needs must be set. There are different ways to set the capacity (the available licenses) in Univa Grid Engine:

1. Consumable-only counter: Set the capacity (the maximum number of available licenses) in the execution host configuration (see `man host_conf`). For site licenses, set the global host. For node locked licenses, set the specific execution host.

Univa Grid Engine keeps track of the licenses in use by jobs. No job requesting license(s) is started if any license is not met.

This is the easiest and most precise way of handling licenses, but licenses must only be consumed by Univa Grid Engine jobs. It is not suited for situations with both interactive and batch license use.

2. Using only external values (load sensor): A load sensor is used to report license usage. See man page `sge\_execd(8)` for information about load sensors. Examples for load sensors are in `$SGE_ROOT/util/resources/loadsensors`.

The load sensor is called at regular intervals (load_report_interval configured in the cluster configuration), it queries the license manager for the available number of licenses and reports this number to Univa Grid Engine.

This setup works in clusters with low job throughput and jobs of longer duration. With higher job throughput or frequent interactive license use, it suffers from race conditions:

- When licenses are consumed interactively, it takes some time (the load report interval) until the load sensor reports the license usage.
- When a job is started, a license is not immediately consumed. During this time period, further jobs requesting a license may be started.

3. Combining consumable with load sensor: This setup combines approaches 1 and 2: Univa Grid Engine keeps track of licenses with the consumable counter, and the actual license usage is reported by a load sensor.

The Univa Grid Engine scheduler will take the minimum of internal license booking and load value as the number of available licenses.

With this setup, interactive license usage is taken into account, and license overbooking due to jobs not immediately drawing licenses is avoided.

Interactive license usage is still reported to Univa Grid Engine by the load sensor with some delay, overbooking licenses due to interactive license usage can still occur.

4. Setting the capacity by an external program: A different approach to reducing the time window for race conditions to a minimum is by using an external component to monitor the license usage and dynamically setting the license capacity in the Univa Grid Engine execution host configuration.

One example of such an external component is the Integrating and Utilizing QLICSERVER.

3.5.1 Integrating and Utilizing QLICSERVER

Qlicserver is a utility for monitoring licenses managed by the FLEXlm license manager, and for controlling interactive vs. batch job license usage.

It allows the administrator to configure the number of licenses used by Univa Grid Engine jobs, and sets the number of available licenses by configuring the capacity of Univa Grid Engine consumables.

The qlicserver has been developed by Marc Olesen and published under a

http://creativecommons.org/licenses/by-nc-sa/3.0.

Summary of the steps required to set up the qlicserver:

- download the flex-grid tar.gz from http://olesenm.github.com/flex-grid
- unpack it to a temporary directory
- copy olesen-flex-grid-*/site to *SGE_ROOT*/SGE_CELL
- configure qlicserver (qlicserver.config, qlicserver.limits)
- create the necessary Univa Grid Engine consumables
- startup qlicserver

A detailed installation and configuration guide is available at http://wiki.gridengine.info/wiki/index.php/Olesen-FLEXlm-Integration

3.6 Managing Priorities and Usage Entitlements

The influence of all policy weights on the final priority

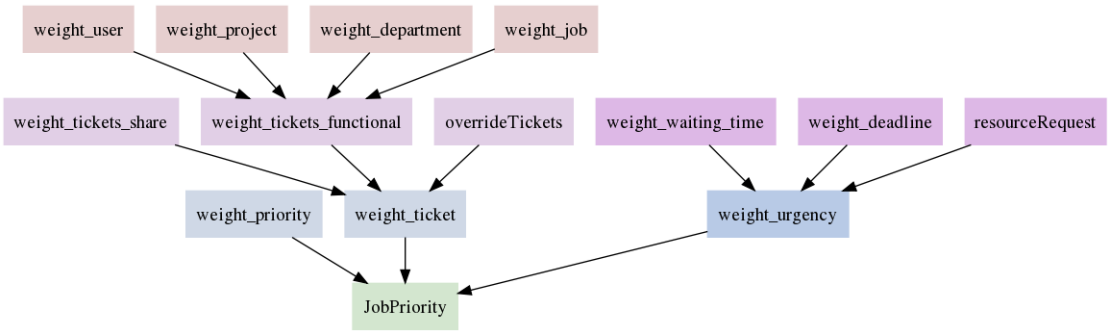


FIGURE: The influences of all policy weights on the final priority

Mapping business rules into cluster use is crucial for companies with a cluster shared by different entities. Univa Grid Engine supports this through different scheduling policies. These policies influence the final order of the job list, which is processed by the scheduler. The scheduler dispatches the jobs to the execution nodes in this order, hence jobs at the top of the list have a greater chance of obtaining resources earlier than jobs at the bottom of the list. Jobs that cannot be dispatched to an execution node due to a lack of resources are deferred to the next scheduling run. This section describes the different policies which influence the job order list. There are three general groups of priorities from which the final priority value is derived: ticket-based priorities, urgency-based priorities, and the POSIX priority. Further information can be found in the `man` pages `sched_conf` and `sge_priority`.

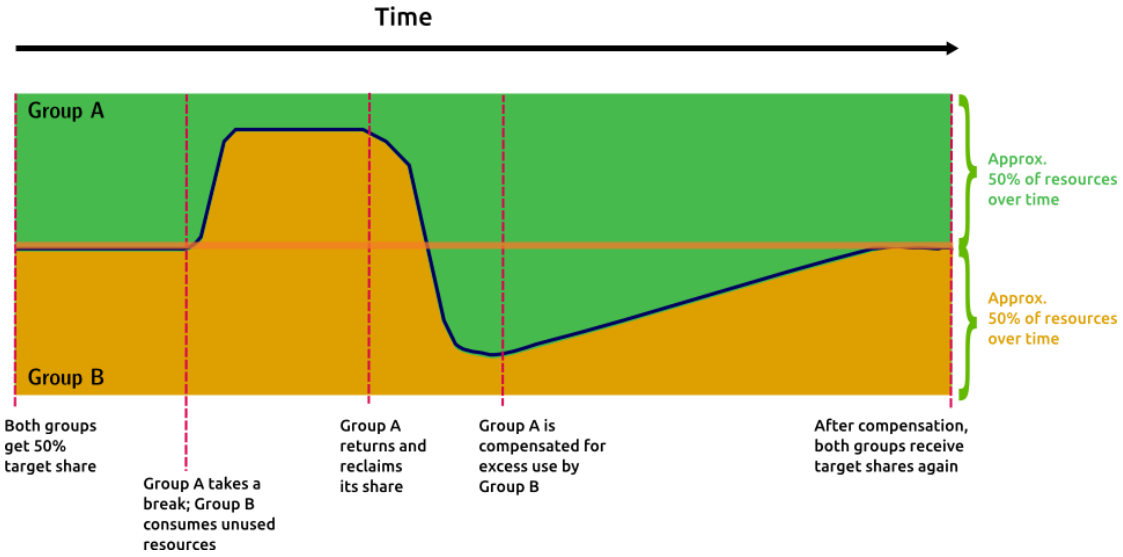
3.6.1 Share Tree (Fair-Share) Ticket Policy

The Univa Grid Engine Share Tree Policy implements fair share scheduling as described in the 1998 paper “Fair Share Scheduler” by J. Kay and P. Lauder published in the Communications of the ACM. The algorithm described in the paper is the basis for fair share implementations across a wide range of computing systems.

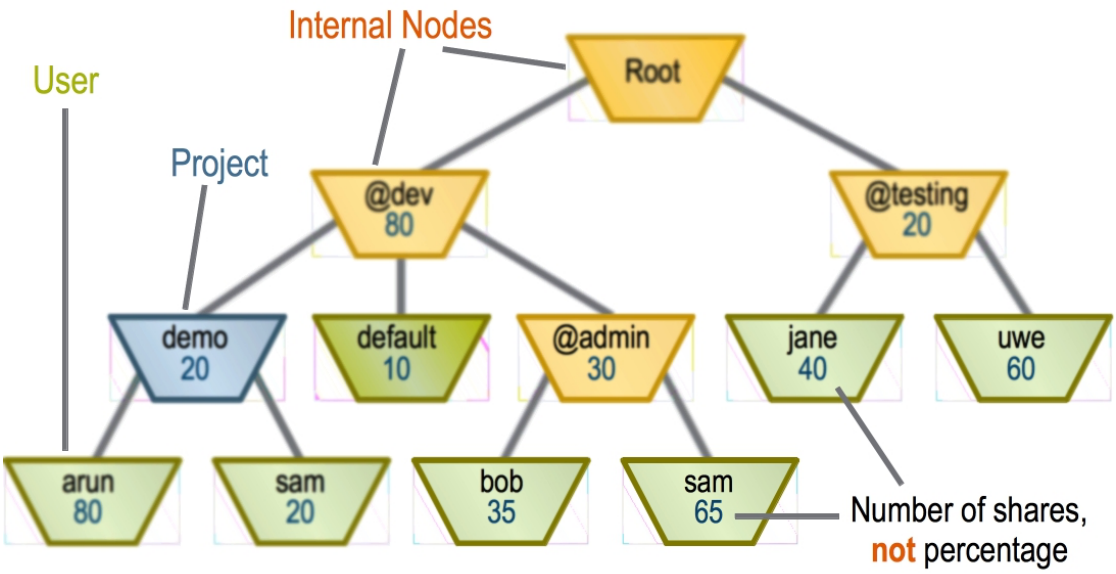
The basic idea of fair share scheduling is to allow an administrator to assign "shares" to users or groups to reflect the percentage of resources which should be granted to that user or group over a period of time. Shares are just numbers which are relative to one another. For example, if we assign 100 shares to Jim and 100 shares to Sally, we are indicating that Jim and Sally should share resources equally. If we assign 200 shares to Sally and 100 shares to Jim, we are indicating that Sally should get twice the resources as Jim.

Rather than explicitly defining a period of time, the share tree algorithm defines a sliding window of time. The sliding window is described by a half-life period. The half-life period is the amount of time after which the resource usage will have decayed to half of the original value. That is, as Jim and Sally use computing resources, the system records and accumulates their resource usage as numeric values such as CPU seconds or the amount of memory (gigabyte-seconds) used. These values are stored by the system and "decayed" based on the half-life period of time. You can also think of the half-life as the amount of time that resource usage will "count against" Jim and Sally. If you want Jim and Sally to share resources equally on a daily basis, the half-life period could be set to 12 hours. If Jim and Sally should share resources equally on a monthly basis, the half-life period could be set for two weeks.

The illustration below shows how resources are shared between two groups. The two groups below could represent two users such as Jim and Sally or could represent two different projects. Initially, if there are jobs submitted by both groups, an equal amount of jobs will be scheduled from each group. If group B receives a greater share of the resources for a period of time, the scheduler will then adjust and attempt to compensate and schedule more jobs from group A. When group A "catches up" the scheduler will begin scheduling the same number of jobs from each group.



Univa Grid Engine provides a sophisticated and flexible implementation of the fair share algorithm. The Share Tree Policy allows supports a hierarchical tree to represent the relationships between users, groups, and projects. The Share Tree Policy also supports fair share scheduling based on a wide variety of resources including CPU, memory, I/O, and consumable resources. These resources can be combined in a flexible way to achieve the sharing goals of a site.



Example Share Tree Configurations

The Share Tree Policy supports a wide variety of sharing configurations. Resource sharing can be specified on a project basis, on a user basis, or on a combined project and user basis. In this section, we list some of the common use cases for share tree configurations and describe how to implement them.

User Share Tree

A common use case is to share equally between all users in the cluster. We accomplish this by defining a special "default" leaf node that will internally expand to represent all users. In this case, we decide that we want the sliding window of time to be about one day. Most jobs are memory bound so we decide to charge usage based on memory.

1. As three different users (Al, Bob, and Carl), submit 100 jobs each sleeping 90 seconds

```
> qconf -clearusage
> qmod -d all.q
> # Al
> for i in `seq 1 100`; do qsub $SGE_ROOT/examples/jobs/sleeper.sh 90; done
> # Bob
> for i in `seq 1 100`; do qsub $SGE_ROOT/examples/jobs/sleeper.sh 90; done
> # Carl
> for i in `seq 1 100`; do qsub $SGE_ROOT/examples/jobs/sleeper.sh 90; done
> qmod -e all.q
```

2. Verify that the jobs are scheduled and running in FIFO (first-in first-out) order.

```
> qstat -u '*' -ext
```

3. Define a share tree with a special "default" leaf node that will internally expand to represent all users.

```
> $ qconf -mstree
id=0
name=ROOT
type=0
shares=1
childnodes=1
id=1
name=default
type=0
shares=1
childnodes=NONE
```

4. Add ENABLE_MEM_DETAILS to support collecting PSS data

```
> qconf -mconf
...
execd_params                 ENABLE_MEM_DETAILS=true
...
```

5. Update the scheduler configuration to schedule based on the share tree by assigning tickets to the share tree policy

```
> qconf -msconf
...
halftime                          12
usage_weight_list                 mempss=1.000000
...
weight_tickets_functional         0
weight_tickets_share              10000
...
weight_ticket                     100.000000
weight_waiting_time               0.000000
weight_deadline                   3600000.000000
weight_urgency                    0.000000
weight_priority                   0.000000
```

6. Verify that the jobs are no longer ordered and running in FIFO (first-in first-out) order.

```
> qstat -u '*' -ext
```

Project Share Tree

A common use case for the Share Tree Policy is the need for two or more projects to share resources. Let's say that our company has two important and busy projects which are funding the shared computing resources equally. We decide we want to grant the computing resources equally to each project. Let's assume also that we decide that the sliding window of time should be about a week. The projects run a variety of jobs, so we decide to charge based on a combination CPU usage and memory usage.

1. Create two projects

```
> qconf -aprj
name projectA
oticket 0
fshare 0
acl NONE
xacl NONE

> qconf -aprj
name projectB
oticket 0
fshare 0
acl NONE
xacl NONE
```

2. Create a share tree with two projects

```
> qconf -mstree
id=0
name=Root
type=1
shares=1
childnodes=1,2
id=1
name=projectA
type=1
shares=100
childnodes=NONE
id=2
name=projectB
type=1
shares=100
childnodes=NONE
```

3. Add ENABLE_MEM_DETAILS to support collecting PSS data

```
> qconf -mconf
...
execd_params                 ENABLE_MEM_DETAILS=true
...
```

4. Create a scheduler configuration to use the share tree policy with a sliding window of about one week

```
> qconf -msconf
...
halftime                          84
```

```
usage_weight_list                 mempss=1.000000,cpu=1.0000
...
weight_tickets_functional         0
weight_tickets_share              10000
...
weight_ticket                     100.000000
weight_waiting_time               0.000000
weight_deadline                   3600000.000000
weight_urgency                    0.000000
weight_priority                   0.000000
...
```

5. Submit jobs in both projects and observe how they share resources

```
> qmod -clearusage
> qmod -d all.q
> for i in `seq 1 20`; do qsub -P projectA $SGE_ROOT/examples/jobs/sleeper.sh 90; done
> for i in `seq 1 20`; do qsub -P projectB $SGE_ROOT/examples/jobs/sleeper.sh 90; done
> qmod -e all.q
```

6. Observe how the jobs and the share tree nodes are sharing resources

```
> $ qstat -ext
...
> $SGE_ROOT/utilbin/$SGE_ARCH/sge_share_mon -a
...
```

Monitoring the Share Tree Policy

Univa Grid Engine includes some tools for monitoring the share tree policy.

qstat

The qstat command can be used to show the results of the share tree policy. One of the best ways to see the results of share tree scheduling is to look at the number of share tree tickets (stckt) granted to each active and pending job. The number of share tree tickets will indicate how the share tree policy is affecting the scheduling order of jobs.

```
> qstat -ext
```

sge_share_mon

The sge_share_mon command was specifically designed to monitor the share tree. The sge_share_mon command reports the following values for each node in the share tree.

Name	Description
curr_time	time stamp of the last status collection for this node
usage_time	time stamp of the last time the usage was updated

Name	Description
node_name	name of the node
user_name	name of the user if this is a user node
project_name	name of the project if this is a project node
shares	number of shares assigned to ths node
job_count	number of active jobs associated to this node
level%	share percentage of this node amongst its siblings
total%	overall share percentage of this node amongst all nodes
long_target_share	long term target share that we are trying to achieve
short_target_share	short term target share that we need to achieve in order to meet the long term target
actual_share	actual share that the node is receiving based on usage
usage	combined and decayed usage for this node
wallclock	accumulated and decayed wallclock time for this node
cpu	accumulated and decayed CPU time for this node
mem	accumulated and decayed memory usage for this node.
io	accumulated and decayed I/O usage for this node
ltwallclock	total accumulated wallclock time for this node
ltcpu	total accumulated CPU time for this node
ltmem	total accumulated memory usage (in gigabyte seconds) for this node
ltio	total accumulated I/O usage for this node

If the -a option is supplied, an alternate format is displayed where the fields in the table above following the usage fields are not displayed. Instead, each node status line contains a field for each usage value defined in the usage_weight_list attribute of the scheduler configuration. The usage fields are displayed in the order that they appear in the usage_weight_list. Below are some of the supported fields.

Name	Description
memvmm	accumulated and decayed memory usage for this node. This represents the amount of virtual memory used by all processes multiplied by the wallclock run-time of the process. The value is expressed in gigabyte seconds.
memrss	accumulated and decayed memory usage for this node. This represents the resident set size (RSS) used by all processes multiplied by the wallclock run-time of the process. The value is expressed in gigabyte seconds. The resident set size is the amount of physical private memory plus the amount of physical shared memory being used by the process.

Name	Description
mempss	accumulated and decayed memory usage for this node. This represents the proportional set size (PSS) used by all processes multiplied by the wallclock run-time of the process. The value is expressed in gigabyte seconds. The proportional set size is the amount of physical private meory plus a proportion of teh shared memory being used by the process.
consumable-resource	accumulated and decayed virtual usage for this node for the consumable resource specified in the usage_weight_list attribute in the scheduler configuration. The amount of the consumable resource which has been requested by the job is multiplied by the wallclock run-time of the job. If the consumable resource is a slot-based resource, the value is also multiplied by the number of slots granted to the job. Memory-type consumable resources are expressed in gigabyte seconds.

Here are some examples of the sge_share_mon command.

1. Show the default share tree nodes (including the header) using the default interval of 15 seconds

```
> $SGE_ROOT/utilbin/$SGE_ARCH/sge_share_mon -h
```

2. Display the share tree leaf nodes including the configured usage names using name=value format

```
> $SGE_ROOT/utilbin/$SGE_ARCH/sge_share_mon -a -n -c 1 -x
```

Advanced Capabilities

Several advanced capabilities have been added to Univa Grid Engine 8.5.0 and later. These are described below.

Share Tree based on Proportional Set Size (PSS) on Linux Systems

The Share Tree Policy now supports scheduling based on Proportional Set Size (PSS) or Resident Set Size (RSS) on Linux Systems. To schedule based on PSS, use the following global and scheduler configuration.

1. Add ENABLE_MEM_DETAILS=true to the execd_params in the global host configuration

```
> qconf -mconf
...
execd_params                 ENABLE_MEM_DETAILS=true
...
```

2. Add “mempss” or “memrss” to the usage_weight_list attribute in the scheduler configuration

```
> qconf -msconf
...
usage_weight_list                 mempss=1.000000
...
```

Share Tree based on consumable resources

The Share Tree Policy now supports scheduling based on consumable resources. Since consumable resources do not generate usage, the scheduler will create virtual usage for jobs which either request a consumable resource or for jobs which receive a default value according to the complex configuration. A use case for using a consumable resource is a site which has a consumable resource called estmem which represents "estimated memory". In this use case, each job submitted is requests a certain amount of estimated memory (qsub -l estmem=1G ...). Each queue or host is configured with an amount of estimated memory. The share tree is configured to schedule based on estimated memory usage.

Here is an example of how to configure the share tree for the estimated memory consumable resource.

1. Create estimated memory consumable resource called "estmem"

```
> qconf -mc
#name shortcut   type relop requestable consumable default  urgency aapre
#------------------------------------------------------------------------
...
estmem    em    MEMORY   <=    YES         YES         1G         0      YES
...
```

2. Add "estmem" to the usage_weight_list attribute in the scheduler configuration. This will cause the scheduler to create virtual usage and to use "estmem" usage for share tree scheduling decisions.

```
> qconf -msconf
...
usage_weight_list                 estmem=1.000000
...
```

3. To monitor the estimated memory usage in the share tree, use the new -a option on sge_share_mon

```
> $SGE_ROOT/utilbin/$SGE_ARCH/sge_share_mon -a
```

Applying Share Tree policy based on slots instead of jobs

In order to reach the target sharing ratios defined in the share tree, the scheduler attempts to balance the active jobs to push users and projects toward their sharing targets. For example, if two projects have pending jobs, and projectA is supposed to get 75% of the resources and projectB is supposed to get 25% of the resources, the scheduler will try to schedule 3 projectA jobs for every one projectB job. However, when scheduling a mix of sequential and parallel jobs

(each using a different number of slots), this method will not likely produce the desired results. The previous Share Tree Policy algorithm did not take into account slot use which means that if a mix of parallel and serial jobs were running or queued, the number of tickets granted to pending jobs would not result in the correct run-time sharing ratios and the share tree targets would not be met. When the scheduler configuration params attribute SHARE_BASED_ON_SLOTS is set to 1 or TRUE, the scheduler will now consider the number of slots being used by running jobs and by pending jobs when pushing users and projects toward their sharing targets as defined by the share tree. That is, a parallel job using 4 slots will be considered to be equal to 4 serial jobs. When the parameter is set to FALSE (default), every job is considered equal. The urgency_slots PE attribute in sge_pe(5) will be used to determine the number of slots when a job is submitted with a PE range.

To turn on sharing based on slots, add the SHARE_BASED_ON_SLOTS to scheduler configuration params attribute.

```
> qconf -msconf
...
params SHARE_BASED_ON_SLOTS=true
```

To demonstrate sharing based on slots versus sharing based on jobs.

1. Create projects “a” and “b”

```
> qconf -aprj
name a
oticket 0
fshare 0
acl NONE
xacl NONE
> qconf -aprj
name b
oticket 0
fshare 0
acl NONE
xacl NONE
```

2. Create or modify parallel environment OpenMP

```
> qconf -ap OpenMP
pe_name               OpenMP
slots                 0
user_lists            NONE
xuser_lists           NONE
start_proc_args       NONE
stop_proc_args        NONE
allocation_rule       $pe_slots
control_slaves        FALSE
job_is_first_task     TRUE
urgency_slots         min
```

```
accounting_summary     FALSE
daemon_forks_slaves    FALSE
master_forks_slaves    FALSE
```

3. Create project share tree with equal projects “a” and “b”

```
> vi sharetree
id=0
name=Root
type=1
shares=1
childnodes=1,2
id=1
name=a
type=1
shares=100
childnodes=NONE
id=2
name=b
type=0
shares=100
childnodes=NONE

> qconf -Astree sharetree
```

4. Configure share tree policy

```
> qconf -msconf
...
job_load_adjustments NONE
...
params SHARE_BASED_ON_SLOTS=true
...
halftime 12
...
weight_tickets_share 10000
...
weight_ticket 100.000000
weight_waiting_time 0.000000
weight_deadline 3600000.000000
weight_urgency 0.000000
weight_priority 0.000000
...
```

5. Configure queue with 16 slots

```
> qconf -mq all.q
...
slots 1,[davidson-desktop=16]
...
```

6. Disable queue and delete any existing jobs

```
> qmod -d all.q
> qdel -u `whoami`
```

7. Clear the share tree usage

```
> qconf -clearusage
```

8. Submit multiple 8 slot jobs under the “a” project

```
> for i in `seq 1 8`; do qsub -P a -pe OpenMP 4 $SGE_ROOT/examples/jobs/sleeper.sh 90; done
```

9. Submit multiple 1 slot jobs under the “b” project

```
> for i in `seq 1 32`; do qsub -P b -pe OpenMP 1 $SGE_ROOT/examples/jobs/sleeper.sh 90; don
```

10. Show pending jobs. Notice the order of the pending jobs contains an equal number of jobs for project “a” and “b”.

```
> qstat -ext | head -16
```

11. Configure sharing based on slots by adding SHARE_BASED_ON_SLOTS=true to the scheduler configuration params attribute.

```
> $ qconf -msconf
...
params SHARE_BASED_ON_SLOTS=true
...
```

12. Enable the queue and show the running and pending jobs. Notice that 2 jobs from project “a” get started (using 8 slots) and 8 jobs from “b” get started (using 8 slots) resulting in equal sharing of the resources.

```
> $ qmod -e all.q
> qstat -ext | head -16
```

Features

Occasionally it might be useful to complete reset the usage for all users and projects back to zero. This can be done using the clear usage command.

```
> qconf -clearusage
```

The long term usage which is displayed by the sge_share_mon program can also be cleared using the clear long term usage command.

```
> qconf -clearltusage
```

3.6.2 Functional Ticket Policy

The functional policy is derived in each scheduler run from scratch and does not incorporate any historical data. It is based on four entities: the submitting **user**, the **job** itself, the **project** in which the job is running, and the **department** of the user. Each of these is assigned an arbitrary weight in order to map the desired business rules into the functional policy.

The policy is turned on in the scheduler configuration with setting `weight_tickets_functional` to an high value. The value determines how many tickets are distributed.

```
> qconf -msconf
...
weight_tickets_functional     1000
```

The relative weight of all entities is configured through the weight values `weight_user`, `weight_project`, `weight_department`, and `weight_job`, which must add up to 1. Because ticket calculation takes time in a scheduling run, the number of jobs considered for the functional ticket policy can be limited with the `max_functional_jobs_to_schedule` parameter. The `share_functional_shares` parameter determines if each job entitled to functional shares receive the full number of tickets or if the tickets are distributed amongst the jobs.

The shares can be configured in the Univa Grid Engine objects itself. In the following example, the shares of two projects are modified in a way that mytestproject receives 70 shares and mytestproject2 receives 30 shares.

```
> qconf -mprj mytestproject
name mytestproject
...
fshare 70
...

> qconf -mprj mytestproject2
name mytestproject
...
fshare 30
...
```

The share of the user is modified similarly through `qconf -mu <username>` with adapting `fshare` there. Departments are a special form of user access lists, with the ability to specify functional shares. They can be modified through the `qconf -mu <departmentname>` command. Job shares are assigned at the time of submission with the `-js qsub` parameter.

In the case where there is more than one job per user in the pending job list at time of scheduling, the full number of calculated tickets is just available to the first job of a user. The second job receives just 1/2 of the number of tickets, the third 1/3 and the nth job get 1/n of the calculated number of tickets.

3.6.3 Override Ticket Policy

The override ticket policy is very helpful for temporary changes in the overall scheduling behavior. With this policy an administrator can grant extra tickets on the following entities: users, departments, projects and pending jobs. It allows a temporary override of a configured and applied policy like the share tree or functional ticket policy. The advantage is that the other policies don't need to be touched just to obtain precedence for special users or projects for a short amount of time.

The override tickets are directly set in the objects such as in the functional ticket policy with the difference that the attribute value is named `oticket`. For granting the extra tickets on job level the pending jobs can be altered with the `-ot <ticketamount>` option.

Example:

The pending job list with granted tickets from the functional policy looks like the following:

```
> qstat -ext -u \*
job-ID  prior   ntckts  name   user      project      ... tckts ovrts otckt ftckt stckt ...
-------------------------------------------------------------------------------------------
   620 1.00000 1.00000 sleep  daniel   mytestproject2 ...  1000     0     0  1000     0
   615 0.45000 0.45000 sleep  daniel   mytestproject2 ...   450     0     0   450     0
   616 0.30000 0.30000 sleep  daniel   mytestproject2 ...   300     0     0   300     0
   617 0.22500 0.22500 sleep  daniel   mytestproject2 ...   225     0     0   225     0
   618 0.18000 0.18000 sleep  daniel   mytestproject2 ...   180     0     0   180     0
   619 0.15000 0.15000 sleep  daniel   mytestproject2 ...   150     0     0   150     0
```

All jobs have just functional project tickets (900), while job 620 has an additional 100 functional user tickets. 620 is the highest priority job hence it gets all 1000 (900 + 100) tickets. The second job of the user has just project tickets (900), but because it is job 2 and receives 1/2 of the tickets (450). Job 616 receives a third (300) and so on.

Now job 619 with total number of 150 tickets (**tckts**) is boosted with by the override policy by adding 1000 override tickets. This can only be done by the operator or administrator of the cluster:

```
> qalter -ot 1000 619
admin@host1 sets override tickets of job 619 to 1000
```

```
> qstat -ext -u \*
job-ID  prior   ntckts  name   user        project     ... tckts ovrts otckt ftckt stckt ...
-------------------------------------------------------------------------------------------
619 0.15000 0.15000 sleep  daniel    mytestproject2 ... 1900  1000  1000   900     0
620 1.00000 1.00000 sleep  daniel    mytestproject2 ...  550     0     0   550     0
615 0.45000 0.45000 sleep  daniel    mytestproject2 ...  300     0     0   300     0
616 0.30000 0.30000 sleep  daniel    mytestproject2 ...  225     0     0   225     0
617 0.22500 0.22500 sleep  daniel    mytestproject2 ...  180     0     0   180     0
618 0.18000 0.18000 sleep  daniel    mytestproject2 ...  150     0     0   150     0
```

The job receives 1000 tickets of the override policy and additionally 900 by the functional ticket policy (project tickets). They are now fully counted because the job is of the highest priority for the user. Job 620 receives 100 functional user tickets and 450 functional project tickets. The rest of the jobs have just functional project tickets.

3.6.4 Handling of Array Jobs with the Ticket Policies

For array jobs a single ticket value is calculated for the whole job. When calculating the overall priority of an array task for sorting the job list before dispatching jobs this per job ticket value is used.

Example:

```
> qstat -g d -ext
job-ID     prior   ntckts  name       ... state ... tckts ...slots ja-task-ID
--------------------------------------...-------...-------...------------
3000000003 0.00000 0.00000 sleep      ... qw    ... 2500  ...    1 1
3000000003 0.00000 0.00000 sleep      ... qw    ... 2500  ...    1 2
3000000003 0.00000 0.00000 sleep      ... qw    ... 2500  ...    1 3
```

Array tasks that have already been running and became pending again (e.g. as they were rescheduled) are treated individually and their number of tickets can differ from the jobs ticket count.

Example:

```
> qstat -g d -ext
job-ID     prior   ntckts  name       ... state ... tckts ...slots ja-task-ID
--------------------------------------...-------...-------...------------
3000000003 0.55976 1.00000 sleep      ... Rq    ... 2500  ...    1 1
3000000003 0.55101 0.12500 sleep      ... qw    ... 1250  ...    1 2
3000000003 0.55101 0.12500 sleep      ... qw    ... 1250  ...    1 3
```

With the global configuration qmaster_params MIN_PENDING_ENROLLED_TASKS the number of array tasks that get treated individually during ticket calculation can be defined.

Valid settings for MIN_PENDING_ENROLLED_TASKS are:

- 0: no pending array task will be enrolled (will be treated individually from the job)
- a positive number: this number of pending array tasks will be enrolled
- -1: all pending array tasks will be enrolled

Example with MIN_PENDING_ENROLLED_TASKS=5

```
> qstat -g d -ext
job-ID     prior   ntckts  name       ... state ... tckts ...slots ja-task-ID
--------------------------------------...-------...-------...------------
```

```
3000000004 0.00000 0.00000 sleep      ... qw    ... 2500  ...    1 1
3000000004 0.00000 0.00000 sleep      ... qw    ... 1250  ...    1 2
3000000004 0.00000 0.00000 sleep      ... qw    ...  833  ...    1 3
3000000004 0.00000 0.00000 sleep      ... qw    ...  625  ...    1 4
3000000004 0.00000 0.00000 sleep      ... qw    ...  500  ...    1 5
3000000004 0.00000 0.00000 sleep      ... qw    ...  416  ...    1 6
3000000004 0.00000 0.00000 sleep      ... qw    ...  416  ...    1 7
```

3.6.5 Urgency Policy

The urgency policies can be distinguished in two groups, depending if the urgency is time or resource based. The time based urgency are the **wait time urgency** and the **deadline urgency**. In Univa Grid Engine there is just one: a very flexible resource-based urgency.

Wait Time Urgency

Most computer resources tend to be occupied thereby forcing low priority jobs to remain on the pending job list (in 'qw' state). While this is the desired behaviour due to other policy configuration, the problem of job starvation can arise. The **wait time urgency** addresses this problem by adding priority to jobs over time. This mean that the priority of a job can be increased relative to the length of time it has spent on the job pending queue.

The wait time urgency is configured in the scheduler configuration:

```
> qconf -ssconf
...
weight_ticket                     0.010000
**weight_waiting_time**           0.000000
weight_deadline             3600000.000000
weight_urgency                    0.100000
weight_priority                   1.000000
```

The relevance can be adjusted relatively to the result of all ticket policies in combination (`weight_ticket`), the deadline policy (`weight_deadline`), the urgency policy (`weight_urgency`), and the POSIX priority (`weight_priority`).

Deadline Urgency

The deadline urgency comes into play when jobs are submitted with a special deadline (`qconf -dl`). The deadline denotes the last point in time when the job should be scheduled. Hence the urgency grows from the time of submission until that time continuously. In order to submit jobs with a deadline the user must be configured in the `deadlineusers` list. The reason for this is to prevent the abuse of this functionality by certain unauthorized users. The weight of the urgency itself is configured in the scheduler configuration:

```
> qconf -ssconf
...
weight_ticket                     0.010000
weight_waiting_time               0.000000
**weight_deadline**         3600000.000000
```

```
weight_urgency                  0.100000
weight_priority                 0.000000
```

The high value is grounded in the calculation of the deadline contribution value:

```
`deadline contribution = max( weight_deadline /`
          `seconds till deadline is reached , weight_deadline)`
```

When the deadline is missed the `weight_deadline` is taken in contribution value. The rapid increase of this value prioritizes those jobs with the most pressing deadlines.

Example

In the following example, a user is added to the `deadlineusers` list. Afterwards 3 jobs are submitted, one without a deadline and the second with a deadline a few minutes in the future, and the third with a few hours in the future.

```
> qconf -mu deadlineusers
name    deadlineusers
type    ACL
fshare  0
oticket 0
entries daniel

> qsub -b y  sleep 100
Your job 33 ("sleep") has been submitted

> qsub -b y -dl 201105041410 sleep 100
Your job 34 ("sleep") has been submitted

> qsub -b y -dl 201105050000 sleep 100
Your job 35 ("sleep") has been submitted
```

The urgency can be viewed with the `qstat` parameter `-urg`:

```
> qstat -urg
job-ID  prior  nurg   urg   rrcontr wtcontr dlcontr name  ...  submit/start at  deadline   ...
-------------------------------------------------------------------------------------------
   34 0.60500 1.00000  12215  1000     0    11215 sleep  ...  05/04/2011 14:04:17 05/04/2011
                                                                                  14:10:00
   35 0.50590 0.00899   1101  1000     0      101 sleep  ...  05/04/2011 14:04:35 05/05/2011
                                                                                  00:00:00
   33 0.50500 0.00000   1000  1000     0        0 sleep  ...  05/04/2011 14:04:03
```

After a few seconds, the different growths of the deadline contribution can be seen:

```
> qstat -urg
job-ID  prior   nurg  urg   rrcontr wtcontr dlcontr name ... submit/start at      deadline
```

```
-----------------------------------------------------------------------------------------
   34 0.60500 1.00000 24841  1000     0      23841 sleep ...  05/04/2011 14:04:17 05/04/2011
                                                                                  14:10:00
   35 0.50542 0.00425  1101  1000     0        101 sleep ...  05/04/2011 14:04:35 05/05/2011
                                                                                  00:00:00
   33 0.50500 0.00000  1000  1000     0          0 sleep ...  05/04/2011 14:04:03
```

Resource-Dependent Urgencies

With resource-dependent urgencies it is possible to prioritize jobs depending on the resources (complexes) that are requested. Sometimes it is desirable to have valuable resources always occupied while cheaper resources remain unused for a specific time. Therefore jobs requesting the valuable resources may obtain these urgencies in order to get a higher position in the scheduler list. The priority of a resource is defined in the last column of the complex configuration:

```
> qconf -mc
#name               shortcut   type        relop requestable consumable default  **urgency**
#-------------------------------------------------------------------------------------------
arch                a          RESTRING    ##    YES         NO         NONE     0
calendar            c          RESTRING    ##    YES         NO         NONE     0
...
slots               s          INT         <=    YES         YES        1        1000
```

As shown, the `slots` complex has an urgency of 1000 while all other resources have an urgency of 0 in a default configuration. The reason why `slots` has a pre-defined urgency is that it is more difficult for parallel jobs, which require more slots, to have requests filled than it is for sequential jobs. The urgency value is taken into account for a job only when it requests it as a **hard** resource request (in difference to a soft resource request).

The weight is again configured in the scheduler configuration:

```
> qconf -ssconf
...
weight_ticket                   0.010000
weight_waiting_time             0.000000
weight_deadline           3600000.000000
**weight_urgency**              0.100000
weight_priority                 1.000000
```

Fair Urgency

Fair Urgency is an extension to the resource-dependent urgencies.

It can be used to achieve an even distribution of jobs over multiple resources.

Example:

- In a cluster there are multiple file servers providing access to multiple file systems.
- Jobs are accessing data from specific file systems.

- File system utilization shall get balanced to optimize file system throughput.

Fair urgency works on resource urgency. Resource urgency is configured by assigning urgency to complex attributes.

For the example scenario 3 complex attributes are created, each with an urgency of 1000.

```
$ qconf -mc
#name               shortcut   type        relop requestable consumable default  urgency
#--------------------------------------------------------------------------------------------
filesystem_1        fs1        BOOL        ##    YES         NO         0        1000
filesystem_2        fs2        BOOL        ##    YES         NO         0        1000
filesystem_3        fs3        BOOL        ##    YES         NO         0        1000
```

As all 3 complex attributes have the same urgency, jobs requesting these resources would all get the same urgency. Assuming no other policies are in place, jobs would get scheduled in the order of submission.

Fair urgency is enabled by listing the resources for which fair urgency scheduling shall be done in the scheduler configuration, attribute fair_urgency_list:

```
qconf -msconf
...
fair_urgency_list                fs1,fs2,fs3
...
```

For testing fair urgency we submit jobs requesting the file system resources, 10 jobs requesting a certain resource each:

```
qsub -l fs1 -N fs1 <job_script>
... (repeat 10 times)
qsub -l fs2 -N fs2 <job_script>
... (repeat 10 times)
qsub -l fs3 -N fs3 <job_script>
... (repeat 10 times)
```

Due to fair urgency being active the jobs are not executed in order of submission, but are interleaved by file system:

```
 qstat
job-ID  prior   name   user     state   submit/start at   queue jclass  slots ja-task-ID
-----------------------------------------------------------------------------------------
     31 0.60500 fs1    sgetest1   qw    05/18/2012 12:06:14                1
     41 0.60500 fs2    sgetest1   qw    05/18/2012 12:06:24                1
     51 0.60500 fs3    sgetest1   qw    05/18/2012 12:06:33                1
     32 0.54944 fs1    sgetest1   qw    05/18/2012 12:06:15                1
     42 0.54944 fs2    sgetest1   qw    05/18/2012 12:06:25                1
     52 0.54944 fs3    sgetest1   qw    05/18/2012 12:06:34                1
```

```
     33 0.53093 fs1   sgetest1   qw    05/18/2012 12:06:16                 1
     43 0.53093 fs2   sgetest1   qw    05/18/2012 12:06:25                 1
     53 0.53093 fs3   sgetest1   qw    05/18/2012 12:06:34                 1
     34 0.52167 fs1   sgetest1   qw    05/18/2012 12:06:16                 1
     44 0.52167 fs2   sgetest1   qw    05/18/2012 12:06:26                 1
     54 0.52167 fs3   sgetest1   qw    05/18/2012 12:06:35                 1
     35 0.51611 fs1   sgetest1   qw    05/18/2012 12:06:17                 1
```

Fair urgency can be combined with other policies, e.g. with the ticket policies.

Example: Functional policy is configured for users to achieve an even balancing of resource usage by users:

Enable functional policy in the scheduler configuration:

```
qconf -msconf
...
weight_tickets_functional         10000
...
```

Give users functional tickets:

```
qconf -muser <user>
fshare 1000
```

As every user submit jobs as shown above (10 jobs per file system).

Jobs are now scheduled to get balancing both on user and on file system:

```
$ qstat -u \*
job-ID  prior name  user  state   submit/start at   queue jclass  slots ja-task-ID
-----------------------------------------------------------------------------------
  31  0.61000 fs1  sgetest1  qw    05/18/2012 12:06:14                 1
  41  0.60045 fs2  sgetest1  qw    05/18/2012 12:06:24                 1
  51  0.60024 fs3  sgetest1  qw    05/18/2012 12:06:33                 1
  61  0.55250 fs1  sgetest2  qw    05/18/2012 12:09:06                 1
  71  0.54795 fs2  sgetest2  qw    05/18/2012 12:09:17                 1
  81  0.54774 fs3  sgetest2  qw    05/18/2012 12:09:25                 1
  32  0.53250 fs1  sgetest1  qw    05/18/2012 12:06:15                 1
  42  0.53042 fs2  sgetest1  qw    05/18/2012 12:06:25                 1
  52  0.53023 fs3  sgetest1  qw    05/18/2012 12:06:34                 1
  62  0.52375 fs1  sgetest2  qw    05/18/2012 12:09:08                 1
  72  0.52167 fs2  sgetest2  qw    05/18/2012 12:09:18                 1
  82  0.52148 fs3  sgetest2  qw    05/18/2012 12:09:26                 1
  33  0.51767 fs1  sgetest1  qw    05/18/2012 12:06:16                 1
```

3.6.6 User Policy: POSIX Policy

The POSIX policy (also called custom policy) is defined per job at the time of job submission. The responsible `qsub` parameter is `-p <value>`. Possible values are those from -1023 to 1024, while values above 0 can be set by the administrator only. This feature is perfect for the user to bring a specific order to his own jobs.

Example

In the following example, several jobs with different priorities are submitted as administrator (which allows also positive priorities).

```
> qsub -b y -p 1 sleep 60
Your job 6 ("sleep") has been submitted

> qsub -b y -p 10 sleep 60
Your job 7 ("sleep") has been submitted

> qsub -b y -p 100 sleep 60
Your job 8 ("sleep") has been submitted

> qsub -b y -p 1000 sleep 60
Your job 9 ("sleep") has been submitted
```

When trying to submit with an invalid priority, the following error message appears:

```
> qsub -b y -p 10000 sleep 60
qsub: invalid priority 10000. must be an integer from -1023 to 1024

> qsub -b y -p -1 sleep 60
Your job 10 ("sleep") has been submitted

> qsub -b y -p -10 sleep 60
Your job 11 ("sleep") has been submitted

> qsub -b y -p -100 sleep 60
Your job 12 ("sleep") has been submitted

> qsub -b y -p -1000 sleep 60
Your job 13 ("sleep") has been submitted
```

The effect of the priorities can be seen with the `qstat` command:

```
> qstat -pri
job-ID  prior   nurg    npprior ntckts **ppri** name   user  state submit/start at    ...
-------------------------------------------------------------------------------------------
     9 1.04328 0.50000 0.98828 0.50000  1000 sleep   daniel   qw    05/05/2011 09:09:47
     8 0.60383 0.50000 0.54883 0.50000   100 sleep   daniel   qw    05/05/2011 09:09:44
     7 0.55988 0.50000 0.50488 0.50000    10 sleep   daniel   qw    05/05/2011 09:09:41
```

```
 6 0.55549 0.50000 0.50049 0.50000     1 sleep   daniel   qw    05/05/2011 09:09:36
 4 0.55500 0.50000 0.50000 0.50000     0 sleep   daniel   qw    05/05/2011 09:09:21
10 0.55451 0.50000 0.49951 0.50000    -1 sleep   daniel   qw    05/05/2011 09:09:58
11 0.55012 0.50000 0.49512 0.50000   -10 sleep   daniel   qw    05/05/2011 09:10:01
12 0.50617 0.50000 0.45117 0.50000  -100 sleep   daniel   qw    05/05/2011 09:10:04
13 0.06672 0.50000 0.01172 0.50000 -1000 sleep   daniel   qw    05/05/2011 09:10:08
```

A job submitted without any priority (job 4) has the priority 0, which results in a normalized priority value (`npprior`) of 0.5. The lower the priority, the lower the normalized value. The absolute weight of the POSIX priority is again defined in the scheduler configuration.

```
> qconf -ssconf
...
weight_ticket                     0.010000
weight_waiting_time               0.000000
weight_deadline             3600000.000000
weight_urgency                    0.100000
weight_priority                   1.000000
```

3.7 Job Placement

Job placement is the main task of the scheduler component within the Univa Grid Engine sge_qmaster process. Job placement is a term for multiple steps that need to be done to correctly decide if and where jobs can be started and which resources they can consume. The rules that the scheduler uses to drive those decisions are defined through manager defined settings in the system.

The scheduler algorithm that is repeatedly triggered is responsible to start new incoming jobs and to handle also all steps required for jobs that want to leave the system because they have been finished. To dispatch new jobs the scheduler has to bring all waiting jobs into an order that represents policies and priorities. Then it has to do the same for hosts and queues that provide resources for jobs and as a final and most important step it has to try to match waiting jobs according to the priority to those hosts/queues, respecting underlying resource requirements and access rights. The two sorting steps for pending jobs and hosts/queues influence the dispatching step that follows afterwards. They can be considered as the major rules to influence the job placement.

Characteristics that influence job sorting of pending jobs are described in more detail in the section Managing Priorities and Usage Entitlements

Subsections following below show the different parts that influence the host/queue sorting and describe therefore how to influence the scheduler which resources of hosts/queues should be consumed first within a cluster.

3.7.1 Host/Queue Sorting

Following general characteristics allow to influence the host sort order:

- Host load

- Affinity of jobs already running on a host (host affinity)

Queue sorting can be influenced by:

- Queue Sequence Number
- Affinity of jobs already running on a queue (queue affinity)
- Outcome of the host sorting (host load and host affinity)

If and to what extend a specific characteristic will be considered by the host/queue sorting depends on a set of weighting factors that are described in more detail in the sections below.

Host Load

The host sorting can be influenced by one or multiple load values that are reported by corresponding hosts. Which load values those are can be defined with the host_sort_formula that allows to weight and combine individual load values and complex values that are defined for all hosts to one final value.

```
host_sort_formula := weighted_value [ operator weighted_value ] .
operator := '+' | '-' .
weighted_value := weight | load_value [ '*' weight ] .
weight := <positive_integer> .
```

load_value represents a load value (see sge_execd(8)) or consumable resource being maintained for each host (see complex(8)) or an administrator defined load value (see the load_sensor parameter in sge_conf(5)). *positive_integer* represents a positive integer value. *np_load_avg* is the default setting for the host_sort_formula.

To what extend the final load value is considered by host sorting can be adjusted with the weight_host_sort parameter in the scheduler configuration.

```
> qconf -ssconf
...
host_sort_formula      np_load_avg
weight_host_affinity   0.0
weight_host_sort       1.0
weight_queue_affinity  0.0
weight_queue_host_sort 1.0
weight_queue_seqno     0.0
...
```

The example above shows an excerpt of a scheduler configuration that enabled host and queue sorting according to the load value np_load_avg. Sorting according to sequence number or due to affinity is disabled. As consequence the scheduler will favor hosts with lower system load to start new jobs.

Affinity

Affinity is an optional value which is assigned to hosts or queue instances based on the running jobs and their resource requests.

Sorting can be done by affinity by setting the weight_host_affinity and/or the weight_queue_affinity weighting factors.

Positive affinity values will will cause hosts/queue instances to be sorted to the top of the host/queue instance list, negative values (anti-affinity) will cause hosts/queue instances to be sorted to the end of the host/queue instance list.

```
> qconf -ssconf
...
weight_host_affinity   1.0
weight_host_sort       0.0
weight_queue_affinity  1.0
weight_queue_host_sort 1.0
weight_queue_seqno     0.0
```

The example above shows an excerpt of a scheduler configuration that enables host and queue sorting according to affinity. Under the assumption that the affinity value of the complex attribute 'slots' has been changed to a positive value (like 1.0) this will cause jobs to build clusters on hosts/queues where already other jobs are running. With a negative value (like -1.0) jobs will fill up hosts and queues evenly.

See also Affinity, Anti-Affinity, Best Fit.

Sequence Number

The sequence number is a manager definable attribute of a queue that will influence the queue sorting when weight_queue_seqno is set accordingly.

```
> qconf -ssconf
...
weight_host_affinity   0.0
weight_host_sort       0.0
weight_queue_affinity  0.0
weight_queue_host_sort 0.0
weight_queue_seqno     1.0
```

The example above shows an excerpt of a scheduler configuration that enables sequence number based scheduling that define clear preference for individual queues residing on a host.

Combining Multiple Placements Policies

It is possible to combine multiple job placement policies by adjusting the weighting factors of each characteristic accordingly:

```
> qconf -ssconf
...
host_sort_formula      np_load_avg
weight_host_affinity   10.0
weight_host_sort       1.0
weight_queue_affinity  10.0
weight_queue_host_sort 1.0
weight_queue_seqno     100.0
```

With the scheme above the the scheduler will utilize queues according to the defined queue sequence number first. If there are multiple queues with the same sequence number then it will utilize those according to queue affinity. Host affinity and host load will have the least impact.

Defaults for All Weighting Factors That Influence Host/Queue Sorting

These are the defaults for all parameters that influence the host/queue sorting:

```
> qconf -ssconf
...
host_sort_formula      np_load_avg
weight_host_affinity   0.0
weight_host_sort       1.0
weight_queue_affinity  0.0
weight_queue_host_sort 1.0
weight_queue_seqno     0.0
```

The influence of affinity and sequence number of queues is disabled as default. Hosts will be sorted according to np_load_avg only.

3.7.2 Affinity, Anti-Affinity, Best Fit

The affinity concept is new since Univa Grid Engine 8.6.0 and it allows to assign each host or queue an affinity value for each consumed resource of jobs that are running on the host or queue.

Affinity can be positive or negative. Positive affinity will attract other pending jobs, negative affinity will reject other pending jobs. Attraction/rejection will work on host and/or queue level if this is enabled by setting the weighting parameters weight_host_affinity and/or weight_queue_affinity.

The affinity values are used in sorting the host list and/or the queue instance list. Sorting based on the affinity value will cause

- affinity (so that jobs build clusters on hosts or queues),
- anti-affinity (so that jobs are distributed on hosts in the cluster or queues residing on hosts)
- or best fit (if a mixture of positive and negative affinity values is defined for different resources)

The affinity value is calculated for every resource (complex value) a running job has requested and which is defined for a host or queue instance either via manual definition in the complex_values of the host or queue instance or via load value.

For all resource types (also non-number based complexes like restring) the absolute number of the affinity complex attribute will be used as affinity for the corresponding resource request of a job. For consumable resources the affinity value of a resource will act as multiplier for the underlying resource requests of a job that are granted.
This means that one *big* running job attracts/rejects to the same extent as *multiple small* running jobs within the same host or queue as long as *big* and *multiple small* consume the same amount of resources.

In case of multiple resource requests of complex attributes with non-zero affinity setting the job's affinity value is the sum of affinity values of corresponding resources.

The sum of affinity values of all jobs already running on a host/queue cause attraction/rejection of corresponding jobs in the pending job list that request the same resources if weight_host_affinity and/or weight_queue_affinity are defined.

```
> qconf -ssconf
...
weight_host_affinity   1.0
weight_host_sort       0.0
weight_queue_affinity  1.0
weight_queue_host_sort 1.0
weight_queue_seqno     0.0
```

The example above shows an excerpt of a scheduler configuration that enables host and queue sorting according to affinity. Under the assumption that the affinity value of the complex attribute 'slots' has been changed to a positive value (like 1.0) this will cause jobs to build clusters on hosts/queues where already other jobs are running. With a negative value (like -1.0) jobs will fill up hosts and queues evenly.

Affinity Use Cases

Affinity Data which is required for running a job is contained on a shared filesystem which is automounted on job start We want to make use of the filesystem caches and make sure that the file system is mounted on as few hosts as possible.

Create a complex attribute for the data source / the filesystem with a positive affinity factor:

```
$ qconf -mc
#name          shortcut  type  relop requestable consumable default  urgency aapre  affinity
filesystem_A   fs_A      BOOL  ==    YES         NO         0        0       NO     1.000000
```

Assign it to the hosts where the data can be made accessible.

```
$ qconf -mattr exechost complex_values filesystem_A=true `qconf -shgrp_resolved @data_hosts`
```

Request the resource when submitting jobs:

```
$ qsub -l filesystem_A ...
```

The actual affinity value of a host / queue instance can be seen in qstat output:

```
$ qstat -F filesystem_A

queuename                      qtype resv/used/tot. np_load  arch          states
---------------------------------------------------------------------------------
all.q@host1                    BIPC  0/2/100        0.00     lx-amd64
        gf:filesystem_A=1 (haff=2.000000)
```

```
3000022205 0.55500 APP_A      user1       r     07/26/2018 17:12:57     1
3000022206 0.55500 APP_A      user1       r     07/26/2018 17:13:07     1
---------------------------------------------------------------------------------
all.q@host2                    BIPC  0/0/10         0.00     sol-amd64
        gf:filesystem_A=1
```

Anti-Affinity A certain application is causing high network load. Therefore we want to distribute jobs running this application over a high number of hosts. We use anti-affinity for this purpose.

Create a complex variable for applications causing high network load:

```
$ qconf -mc
#name          shortcut  type  relop requestable consumable default  urgency aapre  affinity
high_network   hn        BOOL  ==    YES         NO         0        0       NO     -1.000000
```

The jobs can run on any host - we define it on global level:

```
$ qconf -me global
complex_values      high_network=true
```

Request the high_network resource when submitting jobs:

```
$ qsub -l hn ...
```

Anti-Affinity values can be seen in qstat output:

```
$ qstat -F hn
queuename                      qtype resv/used/tot. np_load  arch          states
---------------------------------------------------------------------------------
all.q@host1                    BIPC  0/1/100        0.00     lx-amd64
        gf:high_network=1 (haff=-1.000000)
3000022207 0.55500 HN         user1       r     07/26/2018 17:40:34     1
---------------------------------------------------------------------------------
all.q@host2                    BIPC  0/1/10         0.00     sol-amd64
        gf:high_network=1 (haff=-1.000000)
3000022208 0.55500 HN         user1       r     07/26/2018 17:40:43     1
```

Best Fit Lets assume we have a combination of the two previous examples. Applications need data from a certain filesystem and we want to make uses of filesystem caching, therefore want to use affinity to run jobs on hosts where already jobs requesting the same filesystem are running.

On the other hand one of these applications needing access to the filesystem produces high network load, so we want to distribute them over multiple hosts.

If we combine both **Affinity** and **Anti-Affinity** this is called **Best Fit**.

With the setting done in the two examples above, submit the following jobs:

```
$ qsub -l hn -l fs_A ...
$ qsub -l hn -l fs_A ...
$ qsub -l fs_A ...
$ qsub -l fs_A ...
$ qsub -l fs_A ...
$ qsub -l fs_A ...
$ qsub -l fs_A ...
$ qsub -l fs_A ...
```

This will result in the following resource assignment and affinity values:

```
$ qstat -F fs_A,hn
queuename                      qtype resv/used/tot. np_load  arch          states
---------------------------------------------------------------------------------
all.q@host1                    BIPC  0/1/10         0.00     lx-amd64
        gf:high_network=1 (haff=-1.000000)
        gf:filesystem_A=1 (haff=1.000000)
3000022218 0.55500 APP_A_HN   user1       r     07/27/2018 09:15:35     1
---------------------------------------------------------------------------------
all.q@host2                    BIPC  0/6/10         0.00     lx-amd64
        gf:high_network=1 (haff=-1.000000)
        gf:filesystem_A=1 (haff=6.000000)
3000022219 0.55500 APP_A_HN   user1       r     07/27/2018 09:15:36     1
3000022220 0.55500 APP_A      user1       r     07/27/2018 09:15:46     1
3000022221 0.55500 APP_A      user1       r     07/27/2018 09:15:48     1
3000022222 0.55500 APP_A      user1       r     07/27/2018 09:16:08     1
3000022223 0.55500 APP_A      user1       r     07/27/2018 09:16:09     1
3000022224 0.55500 APP_A      user1       r     07/27/2018 09:16:10     1
---------------------------------------------------------------------------------
all.q@host3                    BIPC  0/0/100        0.07     lx-amd64
        gf:high_network=1
        gf:filesystem_A=1
```

3.8 Advanced Management for Different Types of Workloads

3.8.1 Parallel Environments

Univa Grid Engine supports the execution of shared memory or distributed memory parallel applications. Such parallel applications require some kind of parallel environment.

Examples for such parallel environments are:

- shared memory parallel operating systems
- the distributed memory environments named Message Passing Interface (MPI)
- the distributed memory environments named Parallel Virtual Machine (PVM).

These environments are either provided by hardware vendors or in different forms as open source software. Depending on implementation, its characteristics and requirements, these parallel environments need to be integrated differently as to be used in combination with our software.

Univa Grid Engine provides an adaptive object to integrate parallel environments into the system. The administrator of a Univa Grid Engine system has to deploy such objects with the help of predefined scripts as part of the distribution so that users can easily deploy parallel jobs. Note that the administrator has the ability to:

- define access rules that allow or deny the use of a parallel environment
- define boundary conditions how the resources are consumed within a parallel environment.
- limit access to a parallel environment by reducing the number of available slots or queues

Commands to Configure Parallel Environment Object

To integrate arbitrary parallel environments with Univa Grid Engine it is necessary to define a set of specific parameters and procedures for each. Parallel environment objects can be created, modified or deleted with the following commands.

- `qconf -ap pe_name`

This is the command to add a parallel environment object. It opens an editor and shows the default parameters for a parallel environment. After changing, saving necessary values and closing the editor, a new environment is created.

- `qconf -Ap filename`

Adds a new parallel environment object whose specification is stored in the specified file.

- `qconf -dp pe_name`

Deletes the parallel environment object with the given name.

- `qconf -mp pe_name`

Opens an editor and shows the current specification of the parallel environment with the name `pe_name`. After changing attributes, saving the modifications and closing the editor, the object is modified accordingly.

- `qconf -Mp filename`

Modifies a parallel environment object from file.

- `qconf -sp pe_name`

Shows the current specification of the parallel environment with the name `pe_name`.

- `qconf spl`

Shows the list of names of available parallel environments.

Configuration Parameters of Parallel Environments

Each parallel environment object supports the following set of configuration attributes:

- *Attribute*: `pe_name`
  Value Specification: The name of the parallel environment to be used when attaching it to queues or when administering its definition. This name has be be specified by users that explicitly request a certain type of parallelism for jobs.
- *Attribute*: `slots`
  Value Specification: The number of parallel processes allowed to run in total under the parallel environment concurrently.
- *Attribute*: `user_lists`
  Value Specification: A comma-separated list of user access names. Each user contained in at least one of the enlisted access lists has access to the parallel environment as long as it is not also explicitly excluded via the `xuser_lists` parameter described below.
- *Attribute*: `xuser_lists`
  Value Specification: The `xuser_lists` parameter contains a comma-separated list of user access lists. Each user contained in at least one of the enlisted access lists is not allowed to access the parallel environment. If the `xuser_lists` parameter is set to `NONE` (the default) any user has access. If a user is contained both in an access list enlisted in `xuser_lists` and `user_lists` the user is denied access to the parallel environment.
- *Attribute*: `start_proc_args`
  Value Specification: This parameter defines the command line of a start-up procedure for the parallel environment. The keyword `NONE` can be used to disable the execution of a start-up script. If specified start-up procedure is invoked on the execution machine of the job before executing the job script. Its purpose is it to set up the parallel environment corresponding to its needs. The syntax for the parameter value is: `[username@]path [arg ...]` The optional `username` prefix specifies the user under which this procedure is started. The standard output of the start-up procedure is redirected to the file `NAME.poJID` in the job's working directory, with `NAME` being the name of the job and `JID` which is the job's identification number. Likewise, the standard error output is redirected to `NAME.peJID`. The following special variables expanded at runtime can be used besides any other strings which have to be interpreted by the start and stop procedures to constitute a command line: `$pe_hostfile` The pathname of a file containing a detailed description of the layout of the parallel environment to be set up by the start-up procedure. Each line of the file refers to a host on which parallel processes are to be run. The first entry of each line denotes the host-name, the second entry is the number of parallel processes to be run on the host, the third entry is the name of the queue, and the fourth entry is a processor range to be used when operating with a multiprocessor machine. `$host` The name of the host on which the startup or stop procedures are started. `$job_owner` The user name of the job owner. `$job_id` Univa Grid Engine's unique job identification number. `$job_name` The

name of the job. `$pe` The name of the parallel environment in use. `$pe_slots` Number of slots granted for the job. `$processors` The processor's string as contained in the queue configuration of the primary queue were the parallel job is started (master queue). `$queue` The cluster queue of the queue instance were the parallel job is started.

- *Attribute*: `stop_proc_args`
  Value Specification: The invocation command line of a shutdown procedure for the parallel environment. Similar to `start_proc_args` this method is executed on the execution host. The keyword `NONE` can be used to disable the execution of a shutdown procedure. If specified this procedure is used after the job script has finished. Its purpose is it to stop the parallel environment and to remove it from all participating systems. Syntax, output files and special variables that can be specified are the same as for `start_proc_args`.

- *Attribute*: `allocation_rule`
  Value Specification: The allocation rule is interpreted by the scheduler of the Univa Grid Engine system. This parameter helps the scheduler decide how to distribute parallel processes among the available machines. If, for instance, a parallel environment is built for shared memory applications only, all parallel processes must be assigned to a single machine regardless of how many suitable machines are available. If, however, the parallel environment follows the distributed memory paradigm, an even distribution of processes among machines may be favorable. The current version of the scheduler understands the following allocation rules:

- `$fill_up`

Starting from the best suitable host/queue, all available slots are allocated. Further hosts and queues are filled as long as a job requires slots for parallel tasks.

- `$round_robin`

From all suitable hosts a single slot is allocated until all tasks requested by the parallel job are dispatched. If more tasks are requested than suitable hosts are found, allocation starts again from the first host. The allocation scheme walks through suitable hosts in a best-suited-first order.

- Positive number or `$pe_slots`

An integer number fixing the number of processes per host. If the number is 1, all processes have to reside on different hosts. If the special denominator $pe_slots is used, the full range of processes as specified with the `qsub -pe` has to be allocated on a single host no matter which value belonging to the range is finally chosen for the job to be allocated.

- *Attribute*: `control_slaves`
  Value Specification: This parameter can be set to `TRUE` or `FALSE`. It indicates whether Univa Grid Engine is the creator of the slave tasks of a parallel application on the execution host and thus has full control over all processes in a parallel application, which enables capabilities such as resource limitation and correct accounting. However, to gain control over the slave tasks of a parallel application, a sophisticated parallel environment interface is required, which works closely together with Univa Grid Engine facilities. `FALSE` is the default for this parameter.

- *Attribute*: `job_is_first_task`
  Value Specification: The `job_is_first_task` parameter can be set to `TRUE` or `FALSE`. A value of `TRUE` indicates that the Univa Grid Engine job script will also start one of the tasks of the parallel application, while a value of `FALSE` indicates that the job script and its child processes are not part of the parallel program. In this case the number of slots reserved for the job is the number of slots requested with the `-pe` switch of the submit application plus one additional slot).

- *Attribute*: `urgency_slots`
  Value Specification: For pending jobs with a slot range parallel environment request the number of slots is not determined. This setting specifies the method to be used by Univa Grid Engine to assess the number of slots such jobs might finally get. The following methods are supported:

- Positive number

The specified number is directly used as prospective slot amount.

- `min`

The slot range minimum is used as the prospective slot amount. If no lower bound is specified, range 1 is assumed.

- `max`

The value of the slot range maximum is used as the prospective slot amount. If no upper bound is specified with the range, the absolute maximum possible for the PE's slot setting is assumed.

- `avg`

The average of all numbers occurring within the job's parallel range request is assumed.

- *Attribute*: `accounting_summary`
  Value Specification: This parameter is only checked if `control_slaves` is set to `TRUE`. In this case accounting information is available for every single slave task of a parallel job. These parameters can be set to `TRUE` so that only a single accounting record will be written to the accounting file.

Note that the functionality of the start-up, shutdown procedures is the full responsibility of the administrator configuring the parallel environment. Univa Grid Engine will invoke these procedures and evaluate their exit status. If the procedures do not perform their tasks properly or if the parallel environment or the parallel application behave unexpectedly, Univa Grid Engine has no means of detecting this.

Setup Parallel Environment for PVM Jobs

A central part of the parallel environment integration with Univa Grid Engine is the correct setup of the startup and shutdown procedures. The Univa Grid Engine distribution contains

various script and C program examples that can be used as the starting point for a PVM or MPI integration. These examples are located in the directories `$SGE_ROOT/pvm` and `$SGE_ROOT/mpi`.

Let's have a more detailed look at the startup procedure of the PVM integration. The script is `$SGE_ROOT/pvm/startpvm.sh`. This script requires three command line arguments:

- The first is the path of a file generated by Univa Grid Engine. The content of that file is needed by PVM.
- The second parameter is the host-name of an execution host where the `startpvm.sh` script is started.
- The last parameter is the path of the PVM root directory.

The host file that is created by Univa Grid Engine contains a description of all resources that have been assigned to the parallel job that is in the process of being started. This file has the following format:

- The first entry in each line is a execution host name.
- The second entry defines the number of slots to be available on the host.
- The third entry defines the queue that controls the corresponding available slots.
- The last parameter entry specifies a processor range to be used in case of a multiprocessor machines.

PVM also needs a host file but the file format is slightly different from the default file format generated by Univa Grid Engine. Therefore the `startpvm.sh` script uses the content of the default file to generate one that is PVM specific. After doing this, the script starts the PVM parallel environment. In case this PVM setup has any errors, the `startpvm.sh` script will return with an exit status not equal to zero. Univa Grid Engine will not start the job script in this case and instead indicate an error. If successful, the job script will be started that is now able to use the prepared parallel environment.

A parallel job that has been set up correctly and either finishes or is terminated due a termination request will use the termination method set up in the parallel environment. For the PVM example above, this would mean that the `stoppvm.sh` script is triggered. This script is responsible for halting the parallel environment and terminating processing of the parallel job.

Submitting Parallel Jobs

To run parallel jobs under the control of a certain parallel environment this parallel environment has to be associated with one or multiple queues. Parallel jobs have to request the parallel environment in order to use the needed resources. The queue were the job script of a parallel job is executed is the so called master queue whereas all other queues that provide compute resources for a parallel job are slave queues.

When `job_is_first_task` is set to `FALSE` then the master queue is only used to setup the parallel job. In most cases it will not extensively use the underlying compute resources of the host where the master queue is located. In such setups it might make sense to select a master queue manually with the `-masterq` switch of the `qsub` command to avoid that the job script of the parallel job is started on resources that should be consumed by compute intensive slave tasks of the parallel job.

Attribute	Value specification
`-pe`	`qsub`, `qsh`, `qrsh`, `qlogin` or `qalter` switch is followed by a parallel environment specification of the following format: `pe_name pe_min[-[pe_max]],[-]pe_max` where `pe_name` specifies the parallel environment to instantiate and `pe_min` and `pe_max` specify the minimum or maximum number of slots that might be used by the parallel application.
`-masterq`	This parameter is available for `qsub`, `qsh`, `qrsh` and `qalter` in combination with parallel jobs. It defines or redefines a list of cluster queues, queue domains and queue instances which may be used to become the master queue of the parallel job. The master queue is defined as the queue where the parallel job is started. The other queues to which the parallel job spawns tasks are called slave queues. A parallel job has only one master queue.
`-v` and `-V`	These parameters are available for `qsub` and `qalter`. The `qsh` and `qrsh` support it partly (see qrsh man page). They define or redefine the environment variables to be exported to the execution environment of the job. The same set of variables is also available in the start-up and stop scripts configured in parallel environments.

Table 77: Submit parameters influencing parallel jobs

The following command submits a parallel job:

```
qsub -pe mpi 32-128 \
  -masterq big.q \
  -v SHARED_MEM=TRUE,MODEL_SIZE=HUGE \
  pe_job.sh huge.data
```

- Depending on the definition of the `mpi` parallel environment, the job will use a minimum of 32 slots but a maximum of 128 slots.
- The master queue will be `biq.q`
- Two environment variables are passed with the job. They will be available in the execution context of the job but also in the start-up and stop scripts configured in the `mpi` parallel environment
- The job name is `pe_job.sh` with one parameter `huge.data`

3.8.2 Setting Up Support for Interactive Workloads

To run interactive jobs immediately (see also ‘User Guide -> Interactive Jobs ‘) the executing queue needs to have **interactive** as queue-type.

Set or change queue-type:

```
# qconf -mq <queue_name>
```

INTERACTIVE needs to be added to the **qtype**-line.

Check if *qtype* is **INTERACTIVE**:

```
# qstat -f
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@host1                    IP    0/0/10         0.01     lx-amd64
---------------------------------------------------------------------------------
test.q@host1                   BIPC  0/0/10         0.01     lx-x86
```

qtype has to have "**I**" included.

3.8.3 Setting Up Support for Checkpointing Workloads

Checkpointing is a mechanism that allows a "freeze" of a running application so that it can be restored at a later point in time. This is especially useful for applications that take a long time to complete and when it would be a waste of computer resources to start it from the point at which the application was interrupted (e.g. system crash due to hardware error).

In principle it is possible to distinguish between user level checkpointing and kernel level checkpointing. Kernel level checkpointing must be supported by the underlying operating system where the application is running. If this is the case then applications can be checkpointed without additional effort to rebuild the application. In contrast, user level checkpointing requires some tasks from the author of the application so that it supports checkpointing. The application has to be designed so that the calculation algorithm is able to trigger checkpointing regularly or so that it can be triggered outside the application. Some hardware vendors support this task by providing checkpointing libraries that can be linked against the application code.

Univa Grid Engine does not provide checkpointing for jobs but it does provide the environment in which to integrate jobs already supporting certain levels of checkpointing. The necessary object within Univa Grid Engine is called the checkpointing environment.

Commands to Configure Checkpointing Environments

This checkpointing environment can be set up using the following commands:

- `qconf -ackpt ckpt_name`

This is the command to add a new checkpointing environment. It opens an editor, shows the default parameters for a checkpointing environment. After changing, saving necessary values and closing the editor an new environment is created.

- `qconf -Ackpt filename`

Add a new checkpointing environment whose specification is stored in the specified file.

- `qconf -dckpt ckpt_name`

Deletes the checkpointing environment with the given name.

- `qconf -mckpt ckpt_name`

Opens an editor and shows the current specification of the checkpointing environment with the name `ckpt_name`. After changing attributes, saving the modifications and closing the editor the object is modified accordingly.

- `qconf -Mckpt filename`

Modifies a checkpointing environment from file.

- `qconf -sckpt ckpt_name`

Shows the current specification of the checkpointing environment with the name `ckpt_name`.

- `qconf sckptl`

Shows the list of names of available checkpointing environments.

Configuration Parameters for Checkpointing Environments

Each checkpointing environment supports the following set of configuration attributes:

Checkpointing environment configuration attributes

Attribute	Value specification
`ckpt_name`	The name of the checkpointing environment to be used when attaching it to queues or when administering its definition. This name has to be specified by users that explicitly requests a certain type of checkpointing for jobs.
`interface`	The type of checkpointing to be used. Currently, the following values are supported: `hibernator` The Hibernator kernel level checkpointing is interfaced. `cpr` The SGI kernel level checkpointing is used. `cray-ckpt` The Cray kernel level checkpointing is assumed. `transparent` Univa Grid Engine assumes that jobs that are submitted with reference to this checkpointing environment use a public domain checkpointing environment such as Condor. `userdefined` Jobs submitted with reference to this type of checkpointing interface perform their private checkpointing method. `application-level` In this case all interface commands specified with this object will be used. One exception is the restart_command. Instead of that command the job script itself is restarted.
`ckpt_command`	A command line type command string to be executed by Univa Grid Engine in order to initiate a checkpoint.

Attribute	Value specification
migr_command	A command line type command string to be executed by Univa Grid Engine during a migration of a checkpointing job from one host to another.
restart_command	A command line type command string to be executed by Univa Grid Engine when restarting a previously checkpointed job.
clean_command	A command line type command string to be executed by Univa Grid Engine in order to cleanup after a checkpointing application has finished.
ckpt_dir	A file system location to which checkpoints of potentially considerable size should be stored.
ckpt_signal	A Unix signal to be sent to a job by Univa Grid Engine to initiate a checkpointing generation. The value for this field can either be a symbolic name from the list produced kill -l command or an integer number which must be a valid signal on the system used for checkpointing.
when	Defines the points in time when checkpoints are expected to be generated. Valid values for this parameter are composed by the letters s, m, x and r and any combination thereof without any separating character in between. The same letters are allowed for the qsub -c command which will overwrite the definitions in the checkpointing environment used. The meaning of the letters is defined as follows: s A job is checkpointed, aborted and if possible, migrated if the corresponding sge_execd is shut down where the job is executed. m Checkpoints are generated periodically at the min_cpu_interval defined by the queue in which a job is running. x A job is checkpointed, aborted and if possible migrated as soon as the job gets suspended (manually as well as automatically). r A job will be rescheduled (not checkpointed) when the host on which the job currently runs goes into an unknown state.

Table 78: Checkpointing environment configuration attributes

The Univa Grid Engine distribution contains a set of commands that might be used for the parameters ckpt_command, migr_command or restart_command. These commands are located in the directory $SGE_ROOT/ckpt.

3.8.4 Enabling Reservations

To prevent job starvation, the Univa Grid Engine system has three capabilities: resource reservation, backfilling and advance reservation.

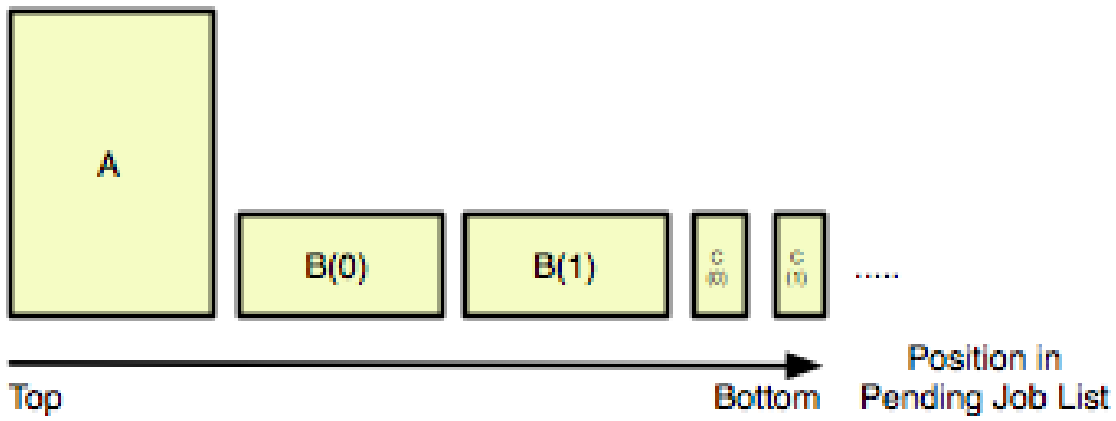
A resource reservation is a job-specific reservation created by the scheduler component for a pending job. During the reservation the resources for jobs of lower priority are blocked so that "job starvation" does not occur.

Advance reservation is a resource reservation completely independent of a particular job that can be requested by a user or administrator and is created by the Univa Grid Engine system. That advance reservation causes the requested resources to be blocked for other jobs that are submitted later on.

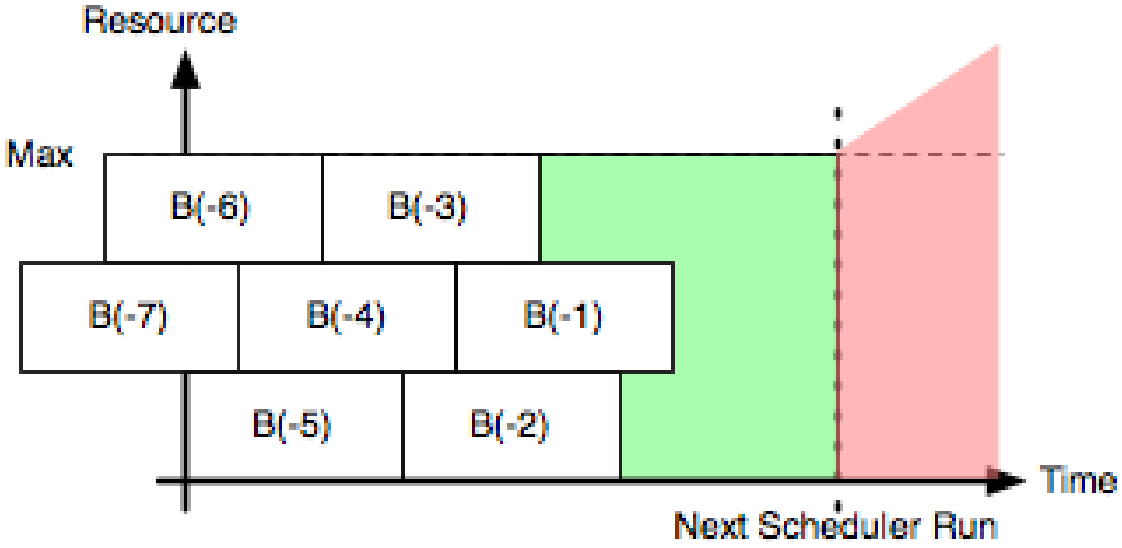
Backfilling is the process of starting jobs of the priority list despite pending jobs of higher priority that might own a future reservation with the same requested resources. Backfilling has only a meaning in the context of resource reservation and advance reservation.

Reservation and Backfilling

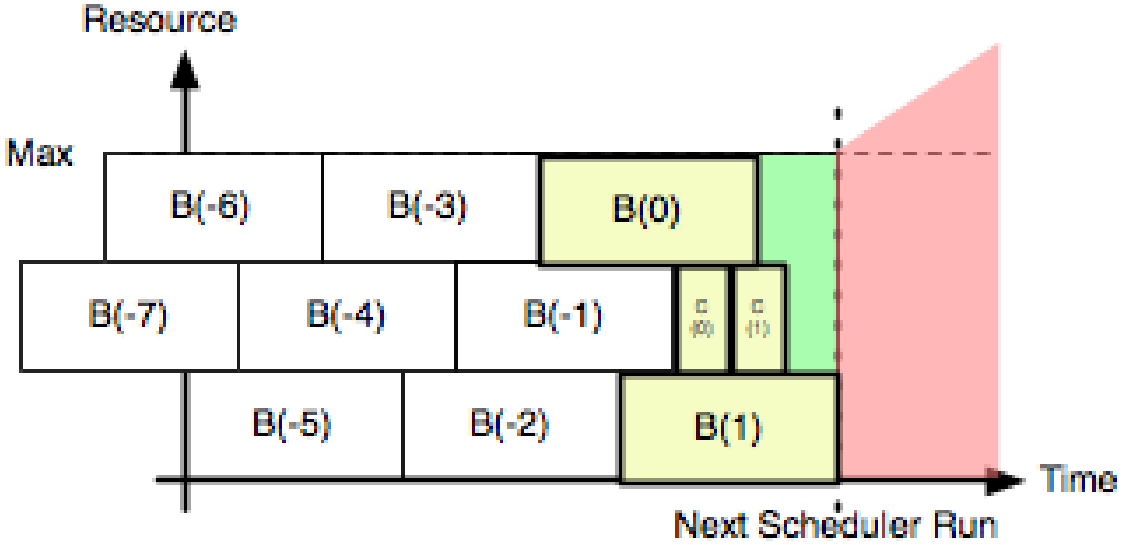
Resource reservation can be used to guarantee resources dedicated to jobs in job priority order. A good example to explain the problem solved with resource reservation and backfilling is the "large parallel job starvation problem". In this scenario there is one pending job of high priority (possibly parallel) named `A` that requires a large quota of a particular resource in addition to a stream of smaller jobs of lower priority `B(i)` and `C(i)` requiring a smaller quota of the same resource.



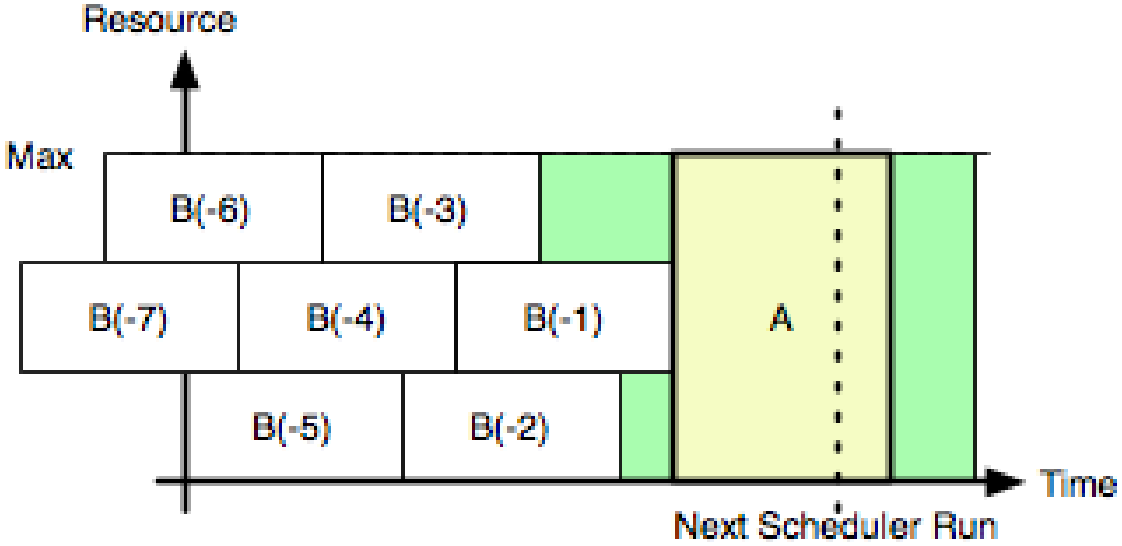
The cluster where the jobs are waiting to be scheduled is already full with running `B(i)` jobs.



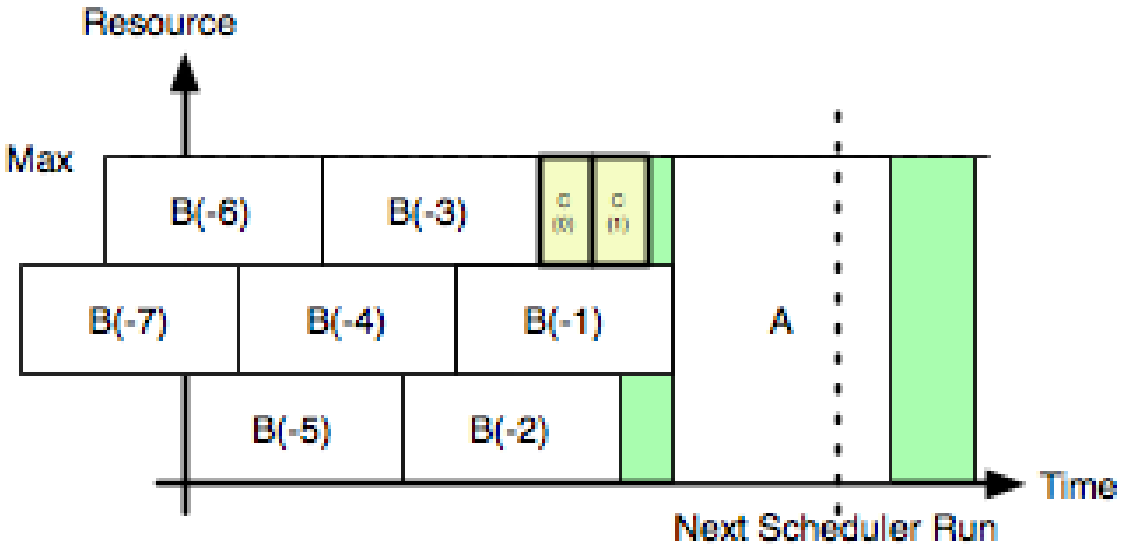
Without resource reservation an assignment for `A` cannot be guaranteed assume the steam of `B(i)` and `C(i)` jobs does not stop - even if job `A` actually has higher priority than the `B(i)` and `C(i)` jobs. Without reservation the scheduler sees only the green area in the resource diagram that is to small for job `A`. The red area (future) is out of the scope of the scheduler. Scheduler without reservation will schedule all lower priority jobs leading to job starvation.



With resource reservation the scheduler's resource planning will be done for the future. Job `A` receives a reservation that blocks lower priority `B(i)` jobs and thus guarantees resources will be available for `A` as soon as possible.



Backfilling allows for the utilization of resources blocked due to a reservation by jobs (and also advance reservations). Backfilling can take place only if there is an executable job with a prospective runtime small enough (like `C(0)` and `C(1)`) to allow the blocked resource be used without endangering the reservation of a job with higher priority. The benefit of backfilling is that of improved resource utilization.



Since resource scheduling requires Univa Grid Engine to look ahead, it is more performance-consuming in reservation mode than in non-reservation mode. In smaller clusters the additional effort is certainly negligible as long as there are only a few pending jobs. With a growing cluster size however and in particular with a growing number of pending jobs, the additional effort makes sense. The key with tuning resource reservations is to determine the overall number that is made during a scheduler interval.

To accomplish this some commandline switches and scheduling parameters are available:

Parameter	Description
`-R`	This submit option is available for `qsub`, `qrsh`, `qsh`, `qlogin` and `qalter`.This option allows the restriction of resource reservation scheduling only to those jobs that are critical. In the example above there is no need to schedule `B(i)` job reservations for the sake of guaranteeing the job `A` resource assignment. The only job that needs to be submitted with the `-R y` is job `A`. This means all `B(i)` jobs can be submitted with the `-R n`. without actually endangering the reservation for `A`. Default is `-R n` if none other is specified.
`-now`	Although it might be requested, reservation is never done for immediate jobs using `-now yes` option.

Table 79: Commandline parameter that influence reservation

Parameter	Description
`max_reservation`	For limiting the absolute number of reservations made during a scheduling interval, the `max_reservation` parameter in the scheduler configuration can be used by Univa Grid Engine administrators. E.g. when `max_reservation` is set to 20 then no more than 20 reservations are made within a scheduling interval and as a result the additional effort for reservation scheduling is limited.
`MONITOR`	If `MONITOR` is added to the `params` parameter in the scheduler configuration then the scheduler records information for each scheduling run allowing for the reproduction of job resource utilization in the file `$SGE_ROOT/$SGE_CELL/common/schedule`.
`DURATION_OFFSET`	If `DURATION_OFFSET` is set then this overrides the default of 60 seconds that is assumed as offset by the Univa Grid Engine scheduler when planning resource utilization as delta between net job runtimes and gross time until resources are available. A job's net runtime as specified with `-l h_rt=...` or `-l s_rt=...` or `default_duration` always differs from job's gross runtime due to delays before and after job start time. Amongst these delays before job start is the time until the end of a `schedule_interval`, the time it takes to deliver a job to `sge\_execd`, the time `prolog` and `starter_method` in queue configuration need and the `start_proc_args` in parallel environments may be affected. The delays after a job's actual run include delays due to a forced job termination (`notify`, `terminate_method` or various checkpointing methods), procedure runs after actual job completion such as `stop_proc_args` in parallel configurations or `epilog` in queues and the delay until a new `schedule_interval`. If the offset is too low, resource reservations can be delayed repeatedly.

Table 80: Parameters in scheduler configuration that influence reservation

3.8.5 Greedy Resource Reservation

Note

Greedy Resource Reservation does not work in combination with manual preemption. Also the integration with Univa License Orchestrator will not be possible for clusters that use Greedy Resource Reservation. Administrators have to take care that corresponding features are disabled before Greedy Resource Reservation is enabled.

The standard resource reservation implementation in Grid Engine requires expensive and extensive analysis in the scheduler. Because of the overhead, it is not practical for use in many large scale clusters with large parallel jobs, where it often can only be enabled for a very small number of reserving jobs, if at all. Turning on standard resource reservation on these systems can extend the scheduling run-times to many minutes and the scheduler may time out. In order to support resource reservation for large scale clusters, a simpler resource reservation has been implemented in Grid Engine called Greedy Resource Reservation (Greedy-RR).

Greedy-RR is a simplified approach to resource reservation where entire hosts are reserved for pending parallel jobs requesting resource reservations. The Greedy-RR examines the pending jobs and reserves enough hosts to allow jobs requesting resource reservations to run. Lower priority jobs which are not reserving resources will not be allowed to use these hosts ensuring that the higher priority jobs with resource reservations are not starved by lower priority jobs. If backfilling is turned on, lower priority jobs that will not delay the start time of the higher priority jobs will be allowed to run. The Greedy-RR algorithm is greedy in that it will attempt to run the highest priority resource reserving job as quickly as possible.

Greedy-RR is low overhead and requires very little scheduling time.

Standard resource reservation in Grid Engine considers the availability of all resources. It builds schedules for every resource and merges these to determine the earliest possible reservation. Greedy-RR is much simpler and only considers the availability of hosts without consideration of all types of consumable and non-consumable resources. Because of this, the Grid Engine administrator should take care to choose parallel environments with hosts which can run jobs requesting resources. It should also be noted that if the execution of jobs at a site tends to be limited more by the consumable resources instead of execution host availability, Greedy-RR may not provide the optimal scheduling.

Several configurable scheduling configuration parameters can be used to control the operation of Greedy-RR.

Parameter	Description
`max_reservation`	Controls the number of jobs for which Greedy-RR will reserve PE host resources.
`backfilling`	Controls if backfilling is allowed for jobs which will not delay the start time of jobs reserving resources.
`greedy_rr`	If this *params* attribute is set to `true`, Greedy-RR will be used instead of standard resource reservation.
`greedy_rr_pe_list`	This *params* attribute is a list of parallel environments or wild-card parallel environments for which hosts will be reserved by Greedy-RR. Multiple entries should be separated by a colon (:). If this attribute is not specified, then Greedy-RR will reserve hosts for all parallel environments.
`greedy_rr_highest_priority`	If this *params* attribute set to `true`, Greedy-RR will consider jobs requesting resource reservations to be the most important jobs and will not allow non-resource reserving jobs to use reserved hosts. Greedy-RR will dynamically partition the system reserving hosts for resource reserving jobs. If this parameter is used, it is also recommended that resource reserving jobs be set to a high priority. The default value is `false`.
`greedy_rr_backfill_non_rr_jobs`	If this *params* attribute set to `true`, Greedy-RR will allow backfilling with non-resource-reserving jobs. If set to `false`, Greedy-RR will only backfill with lower priority resource-reserving jobs. The default value is `false`.

Parameter	Description

The following is a list of important items to consider when using Greedy-RR:

- Only parallel jobs are reserved. Serial jobs are not considered. Array jobs are not supported.
- Entire hosts are reserved. Partial hosts or individual queues cannot be reserved.
- Consumable and non-consumable resources are not taken into consideration when reserving hosts.
- Only jobs submitted with `qsub -R yes` will reserve resources.
- The `qstat -rr` command can be used to view those jobs which have reserved resources.
- The PE allocation rule `$round_robin` should be avoided as it can partially fill up hosts.
- Reasonable backfilling requires job run-times to be specified using `h_rt`, `s_rt`, `d_rt` or setting a `default_duration`)

Advance Reservation

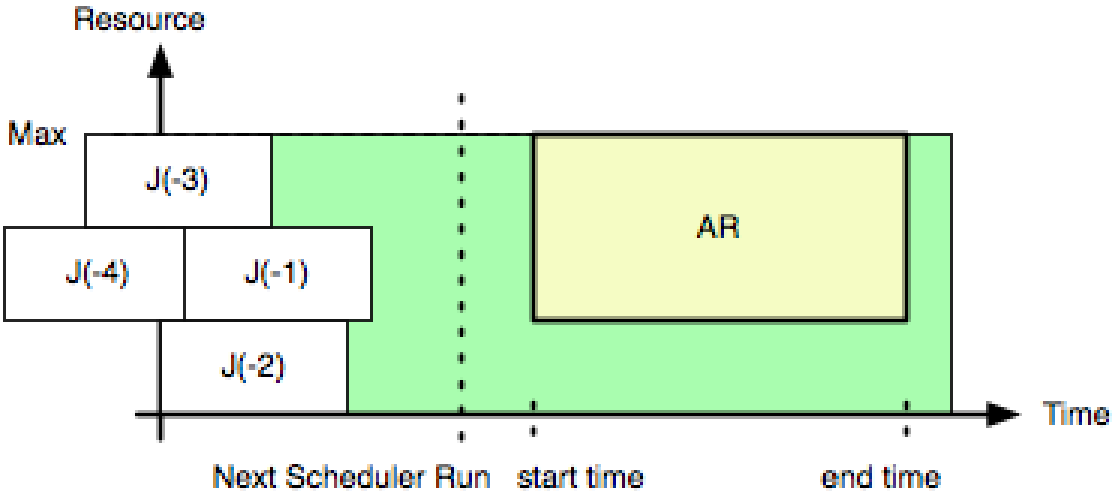
Advance reservations can be compared to the ticket reservation system of a cinema. If a group of people intends to see a specific movie then someone can reserve a defined number of tickets. If the tickets are reserved then people can meet when the movie begins. A seat is available to them during the movie but they must leave when the movie ends so that the theater is available again for the next showing.

An advance reservation is defined as a reservation of resources done by the scheduler due to a user or administrator request. This reservation is made at the beginning independent from a particular job. After it is created, multiple users may submit jobs to advance reservations to use the reserved resources.

Advance reservations have the following characteristics:

- defined start time
- defined duration
- defined set of resources

The absence of one of these characteristics makes it impossible for the Univa Grid Engine scheduler to find the necessary resources so that the advance reservation can be scheduled.



An advance reservation has the following states:

State	Description
`r`	Running. Start time has been reached.
`d`	Deleted. The AR was manually deleted.
`x`	Exited. The end time has been reached.
`E`	Error. The AR became invalid after the start time has been reached
`w`	Waiting. The AR was scheduled but start time has not been reached
`W`	Warning. The AR became invalid because resources that were reserved are not available in the cluster anymore.

Table 82: States of an advance reservation

The following commands address advance reservations or influence them:

- `qrsub`

Used to submit a new advance reservation. Returns the identifier `{ar_id}` that is needed as parameter for other commands.

- `qralter`

Command to modify existing advance reservations.

- `qrdel {ar_id}`

Deletes the advance reservation with ID `{ar_id}`.

- `qrstat`

Command to view the status of advance reservations.

- `qconf -au {username} arusers`

Adds a user with name `{username}` to the access list that allows it to add/change/delete advance reservation objects.

- `qsub -ar {arid}`

Submits a job into a specific advance reservation with the ID `{ar_id}`.

If an AR is submitted with `qrsub`, then the start or end time with the duration of the job must be specified. Other parameters are similar to those of the `qsub` command. Note that only users who are in the `arusers` access list have the right to submit advance reservations.

State	Description
`-a {date_time}`	Defines the activation (start) date and time of the advance reservation. The switch is not mandatory. If omitted, the current date and time is assumed. Either a duration or end date and time must also be specified. Format of `-a {date_time}` is: `[[CC]YY]MMDDhhmm[.ss]` where `CC` denotes the century, `YY` the year, `MM` the month, `DD` the day, `hh` the hour, `mm` the minutes and `ss` the seconds when the job could be started.
`-A {account_string}`	Identifies the account to which the resource reservation of the AR should be charged.
`-ckpt {ckpt_name}`	Selects the checkpointing environment the AR jobs may request. Using this option guarantees that only queues providing this checkpointing environment will be reserved.
`-d {time}`	Defines the duration of the advance reservation. The use of this switch is optional if an end date and time is specified with `-e`
`-e {date_time}`	Defines the end date and time of the advance reservation. The use of this switch is optional if `-d` is requested. Format of `{date_time}` is the same as for `-a`.
`-he {y_or_n}`	Specifies the behavior when the AR goes into an error state. This will happen when a reserved host goes into an unknown state, a queue error happens, or when a queue is disabled or suspended. A hard error means that as long as the AR is in error state jobs requiring the AR will not be scheduled. If a soft error is specified (with `-he y`) then the AR stays usable with the remaining resources. By default, soft error handling is used.
`-l {requests}`	The created AR will provide the given resources specified in `{requests}`. Format of `{requests}` is the same as for `qsub -l`.

State	Description
`-m {occasion}`	Defines under which circumstances mail is sent to the AR owner or to the users defined myth `-M` option. `{occasion}` can be a letter combination of the letters `b`, `e` and `a` or the letter `n` where `b` is a beginning mail, `e` an end mail, `a` a mail when the AR goes into error state and `n` will disable sending of any mail. By default, no mail will be sent.
`-now {y_or_n}`	This option impacts the queue selection for reservation. With `y`, only queues with the type `INTERACTIVE` assigned will be considered for reservation and `n` is default.
`-N {name}`	The name of the advance reservation.
`-pe {pe_name} {pe_range}`	Selects the parallel environment the AR jobs may request, Using this option guarantees the queues providing this parallel environment will be reserved.
`-w {level}`	Specifies the validation level applied to the AR request. `v` does not summit the AR but prints an validation report whereas `e` means that an AR should be rejected if requirements cannot be fulfilled. `e` is the default.

Table 83: `qrsub` command line switches

3.8.6 Simplifying Job Submission Through the Use of Default Requests

Default requests are job requests that are normally specified at time of submission in the form of command line switches and arguments to applications like `qsub`, `qsh` or `qrsh`.

These requests are:

- resource requests of resources that are needed to execute a job successfully (e.g. `-l`, `-pe`)
- descriptions of execution environments to define the context in which jobs are executed (e.g. `-S`, `-shell`, `-notify`)
- certain hints for the scheduler to help identify resources that might be used for execution (e.g. `-q`, `-pe`, `-cal`, `-ckpt`)
- parameters that define the importance of a job compared to other jobs (e.g `-p`, `-js`)
- identifiers that might be used later on for accounting (e.g. `-N`, `-A`) ...

In case of absence of these parameters, additional work is required for the administrator or for the user who discovers that jobs were not started at all or started using resources that are not suitable for the job.

Within Univa Grid Engine it is possible to define default requests to solve that problem. These default requests are used as job requests in the absence of a corresponding request in the submits command line specification.

Locations to set up such default requests are:

```
# the default request file located in `$SGE_ROOT/$SGE_CELL/common`
# the user default request file `.sge_request` located in `$HOME` of the submit user
# the user default request file `.sge_request` located in the current working directory
  where the submit client application is executed.
```

If these files are available then they will be used for every job that is submitted. They are processed in the order mentioned above. After that, the submit options embedded in the job script will be handled as the last switches and parameters that were passed with the command line of the submit application. When during this processing the -clear option is detected, any previous settings are discarded.

The file format of default request files is as follows:

- Blank lines and lines beginning with a hash character (#) are skipped.
- Any other line has to contain one or more summit requests. The requests have the same name and format as they are used with the `qsub` command.

The following is an example of a default request definition file:

```
# Default Requests File

# request a host of architecture sol-sparc64 and a CPU-time of 5hr
-l arch=sol-sparc64,s_cpu=5:0:0

# job is not restarted in case of a system crash
-r n
```

Having defined a default request definition file like this and submitting a job as follows:

```
qsub test.sh
```

would have precisely the same effect as if the job is submitted with:

```
qsub -l arch=sol-sparc64,s_cpu=5:0:0 -r n test.sh
```

3.8.7 Job Submission Verifiers

Job Submission Verifiers (JSVs) are UNIX processes that communicate with Univa Grid Engine components to verify jobs before entering the Univa Grid Engine system. These JSV processes can then decide if Univa Grid Engine should accept a job, modify the job before it is accepted or completely reject a job. Accepted jobs will be put into the list of pending jobs.

The Univa Grid Engine admin user might define JSVs to:

- ensure the accuracy of submitted jobs.

- verify additional things that might be needed during job execution which are out of the scope of Univa Grid Engine like certain access rights to hardware or software.
- inform the user of details of job submission, estimated execution times, cluster policies, ...
- integrate additional software components

Also users submitting jobs can setup JSVs to:

- set up job templates for those jobs that are submitted often.
- ensure that certain environment variables are passed with jobs so that they can be executed successfully.

Using JSVs for Ensuring Correctness of Job Submissions

Univa Grid Engine knows two different JSV types that are named Client JSV and Server JSV. Client and Server JSVs have slightly different characteristics:

Client JSV	Server JSV
Can be defined by users that submit jobs and/or administrators.	Only administrators can define server JSVs.
Always executed on the submit host where the user tries to submit a job.	Server JSV instances are executed on the machine where the sge_qmaster process is running.
Are executed under the submit user account with the environment of the submit user	Either executed as admin user or under an account specified by the administrator.
Client JSVs communicate with the submit client and therefore have the ability to send messages to the stout stream of the corresponding submit client. This is helpful when administrators want to notify submit users about certain conditions.	Server JSVs exchange information with qmaster users about certain conditions. internal threads. Logging can be done to the message file of sge_qmaster process.
They are terminated after one job verification.	Live as long as sge_qmaster process is alive and JSV script does not change.
Has no direct impact on the cluster throughput.	Have to be written carefully. Due to the fact that these JSVs directly communicate with qmaster these JSV type may decrease submission rate and cluster throughput.

Table 84: Client/Server JSV Characteristics

Locations to Enable JSV

To enable the JSV infrastructure, the submit or admin user of a Univa Grid Engine system has to prepare one or multiple script or binary applications. The path to that JSV must be configured

within Univa Grid Engine so that the corresponding application will be triggered when a new job tries to enter the system. In principle it is possible to specify the `-jsv` parameter with various submit clients and the `jsv_url` parameter can be defined within the cluster configuration. This allows the specification of JSV's at the following locations:

```
# `-jsv` used as command line parameter with the submit client
# `-jsv` used in the `.sge_request` file located in the current working directory
    where the submit client is executed
# `-jsv` used in the `.sge_request` file located in `$HOME` of the user that tries
    to submit a job
# `-jsv` used in the global `sge_request` file in `$SGE_ROOT/$SGE_CELL/common`
# `jsv_url` defined in the cluster configuration.
```

If a JSV is defined at one of the five locations, then it will be instantiated and used to verify incoming jobs in a Univa Grid Engine system. JSV 1, 2, 3, and 4 are client JSVs. 5 is a server JSV. The question of how many JSVs are needed in a cluster and where the best location is to set up a JSV depends on the tasks to be achieved by the JSV's. JSV 1, 2, 3 and 4 are started as submit user whereas for JSV 5 the administrator can define the user under which the JSV is executed.

In the extreme case where all configuration locations would be used to set up JSV's, this would result in up to 5 JSV instances. The instance 1 would get the specification of a job as it was defined in the submit client. If it would allow the submission of this job or when the job is accepted with some corrections then the new job specification would be passed to JSV instance 2. Also this JSV would have the capacity to accept or modify the job. The result of each JSV verification or modification process would be passed on to the next instance until JSV 5 either accepts, corrects or rejects the job.

The verification process is aborted as soon as one JSV rejects a job. In this case the submit user will get a corresponding error message. If the job is accepted or corrected then qmaster will accept the job and put it into the list of pending jobs elected to be scheduled later on.

JSV Language Support

JSV processes are started as child processes either from a submit client or the sge_qmaster process. The stdin/stout/stderr channels of a JSV process are connected to the parent process via Unix pipes so that processes can exchange information like job specifications, environments and verification results. Due to this common setup it is possible to write JSVs in any principle programming language.

Perl, TCL, Bourne Shell and Java JSVs are supported out-of-the-box because the Univa Grid Engine distribution contains modules/libraries that implement the necessary communication protocol to exchange data between Univa Grid Engine components and JSVs. The communication protocol is documented so that other language supports may be easily implemented. Note that due to performance reasons it is recommended to write Server JSVs in Perl or TCL (Never use Bourne Shell scripts for production systems. Use it only for evaluation of JSV).

Predefined language modules for the different scripting languages and example JSV scripts can be found in the directory `$SGE_ROOT/dist/util/resources/jsv`. These modules provide functions to perform the following tasks:

- To implement the main loop of the script

- To handle the communication protocol to communicate with Univa Grid Engine components
- To provide access functions to the job submit parameters
- To provide access functions to the job environment specification
- To define reporting functions
- To define logging infrastructure

If these predefined modules are used then only two functions have to be written to create a fully working JSV script.

```
01  #!/bin/sh
02
03  jsv_on_start()
04  {
05      return
06  }
07
08  jsv_on_verify()
09  {
10     jsv_accept "Job is accepted"
11     return
12  }
13
14  . ${SGE_ROOT}/util/resources/jsv/jsv_include.sh
15
16  jsv_main
```

- If this JSV is started, then it will source the predefined Bourne Shell language module (line 14)
- With the call of `jsv_main()` function (line 15) the main loop of the JSV script entered handles the communication protocol which triggers two callback functions when a job verification should be started
- Function `jsv_on_start()` (line 3) is triggered to initiate a job verification. In this function certain things can be initialized or information can be requested from the communication partner. In this example the function simply returns.
- The function `jsv_on_verify()` (line 8) is automatically triggered shortly after `jsv_on_start()` has returned. The time in between those two calls is used to exchange job-related information between client/sge_qmaster and the JSV process.
- In this small example the function `jsv_on_verify()` only accepts the job without further verification. This is done with the function `jsv_accept()` (line 10)
- Note that in case of client JSV, the JSV process terminates shortly after jsv_on_verify() is returned and before the submit client terminates. In case of server JSV, the process remains running since both defined functions will be triggered one after another for each job requiring verification.

JSV Script Interface Functions

This section lists the provided interface functions that are defined in script language modules and that can be used within `jsv_on_start()` and `jsv_on_verify()` to implement job verification.

Accessing Job Parameters

The following functions can be used to access job specification within a JSV that was either defined by the submit environment, the submit client or the switches used in combination with a submit client to submit a job. The `param_name` parameter that has to be passed to those function is a string representing a submit client switch. In most cases the name of `param_name` is the same as the switch name used in combination with `qsub` command. Sometimes multiple `param_name`'s have to be used to retrieve the information in JSV that has been defined at commandline using only one switch. The functions also accept some pseudo-`param_name`'s to find more detailed information about the submit client. A full list of `param_name`'s can be found in one of the following sections.

- `jsv_is_param(param_name)`

Returns whether specific job parameters are available for the job being verified. Either the string `true` or `false` will be returned.

- `jsv_get_param(param_name)`

Returns the value of a specific job parameter. This value for a `param_name` is only available if `jsv_is_param(param_name)` returns true. Otherwise an empty string will be returned.

- `jsv_set_param(param_name, param_value)`

This function changes the job parameter `param_name` to `param_value`. If `param_value` is an empty string then the corresponding job `param_name` will be deleted similar to the function `jsv_del_param(param_name)`. As a result the job parameter is not available since the corresponding command line switch is not specified during job submission. For boolean parameters that only accept the values `yes` and `no` as well as for the parameters `c` and `m` it is not allowed to pass an empty string as `param_value`.

- `jsv_del_param(param_name)`

Deletes the job parameter `param_name` from the job specification as if the corresponding submit switch was not used during submission.

Examples

```
01  jsv_on_verify()
02  {
03     ...
04
05     if [ `jsv_is_param b` = "true" -a `jsv_get_param b` = "y" ]; then
```

```
06        jsv_reject "Binary job is rejected."
07        return
08     fi
09
10     ...
11  }
```

The script above is an excerpt of a `jsv_on_start()` function (Bourne shell):

- The first part of the expression in line 5 tests if the `-b` switch is used during the job submission.
- The second part of the expression tests if the passed parameter is `y`
- If a binary job is submitted then the corresponding job will be rejected in line 6.
- The error message that will be returned by qsub is passed as parameter to `jsv_reject()`

  01 if ["`jsv_get_param pe_name`" != ""]; then 02 slots=`jsv_get_param pe_min` 03 i=`expr $slots % 16` 04
  05 if [$i -gt 0]; then 06 jsv_reject "Parallel job does not request a multiple of 16 slots" 07 return 08 fi 09 fi

The section above might be used in `jsv_on_start()` function (Bourne shell):

- In line 1 it is checked if -pe switch was used at command line
- The `pe_min` value contains the slot specification. If a job was specified e.g with `qsub -pe pe\_name 17` then `pe_min` will have a value of 17.
- Line 3 calculates the remainder of division.
- Line 5 uses this reminder to se if the specified slots value was a multiple of 16.
- The job is rejected in line 6.

Accessing List-Based Job Parameters

Some job parameters are lists that can contain multiple variables with an optional value. Examples for those parameters include job context specifications, resource requests lists and requested queue lists. To access these parameters as well as its values, the following functions have to be used:

- `jsv_sub_is_param(param_name, variable_name)`

This function returns `true` if the job parameter list `param_name` contains a variable with the name `variable_name` or `false` otherwise. `false` might also indicate that the parameter list itself is not available. The function `jsv_is_param(param_name)` can be used to check if the parameter list is not available.

- `jsv_sub_get_param(param_name, variable_name)`

This function returns the value of `variable_name` contained in the parameter list `param_name`. For list elements that have no value, an empty string will be returned as well as for the following `param_name`'s: `hold_jid`, `M`, `masterq`, `q_hard`, `q_soft`. For the `param_name`'s `l_hard` and `l_soft` the value is optional. The absence of a value does not indicate that `variable_name` is not contained in the list. `jsv_is_sub_param(param_name)` can be used to check this.

- `jsv_sub_add_param(param_name, variable_name, variable_value)`

This function adds a `variable_name`/`variable_value` entry into the parameter list `param_name`. If `variable_name` is already contained in that list the corresponding value will be replaced. `variable_value` might be an empty string. For certain `param_name`'s the `variable_value` must be an empty string. Find the list above in the section `jsv_sub_is_param(param_name, variable_name)`

- `jsv_sub_del_param(param_name, variable_name)`

Deletes a variable and if available the corresponding value from the list with the name `param_name`.

Example

```
01  l_hard=`jsv_get_param l_hard`
02  if [ "$l_hard" != "" ]; then
03     has_soft_lic=`jsv_sub_is_param l_hard soft_lic`
04
05     if [ "$has_soft_lic" = "true" ]; then
06        jsv_sub_add_param l_hard h_vmem 4G
07     fi
08  fi
```

- Line 1 returns the value of the `-l` commandline switch.
- If the value for that parameter is not empty then there is at least one resource request passed during job submission.
- Line 3 checks if it contains the `soft_lic` resource request.
- `has_soft_lic` will be set to `true` in this case (line 5).
- If this was specified then `h_vmem` will be set to `4G` (line 6).

Preparing a Job Verification

This function can be used in `jsv_on_start()` to request more detailed information for the job verification process before the verification is started:

- `jsv_send_env()`

This function can only be used in `jsv_on_start()`. If it is used there then the full job environment information will be available in `jsv_on_verify()` for the job that should be verified next. This means that the functions `jsv_is_env()`, `jsv_get_env()`, `jsv_add_env()` and `jsv_mod_env()` can be used within `jsv_on_verify()` to access, modify, delete environment-related information if the job specification that is passed with the `-v` or `-V` switches of the different command line applications. By default, the job environment is not passed to JSVs for performance reasons. Job environments might become big (10K or more). Automatically transferring it for each job would slow down the executing components. Also note that the data in the job environment cannot be verified by Univa Grid Engine and this might therefore contain data which could be misinterpreted in the script environment and cause security issues.

Logging Status

The following JSV logging functions are available.

- `jsv_log_info(message)`

The passed `message` string is transferred to the submit client invoking the executing client JSV or be sent to `sge\_qmaster` process in case of server JSV. Submit clients will then write the `message` to the stout stream of the submit application whereas in case of server JSV `message` is written as info message into the message file of `sge\_qmaster`.

- `jsv_log_warning(message)`

The passed `message` string will be transferred to the submit client that invoked the executing client JSV or it will be sent to `sge\_qmaster` process in case of server JSV. Submit clients will then write the `message` to the stout stream of the submit application whereas in case of server JSV `message` is written as warning message into the message file of `sge\_qmaster`.

- `jsv_log_error(message)`

The passed `message` string will be transferred to the submit client that invoked the executing client JSV or it will be sent to `sge\_qmaster` process in case of server JSV. Submit clients will then write the `message` to the stout stream of the submit application whereas in case of server JSV `message` is written as error message into the message file of `sge\_qmaster`.

Example

```
01  l_hard=`jsv_get_param l_hard`
02  if [ "$l_hard" != "" ]; then
03     context=`jsv_get_param CONTEXT`
04     has_h_vmem=`jsv_sub_is_param l_hard h_vmem`
05
06     if [ "$has_h_vmem" = "true" ]; then
07        jsv_sub_del_param l_hard h_vmem
08        if [ "$context" = "client" ]; then
09           jsv_log_info "h_vmem as hard resource requirement has been deleted"
10        fi
11    fi
12  fi
```

- Line 3 identifies if the JSV is a client or server JSV.
- In case of server JSV (line 8) ...
- ... the JSV prints the log message `"h_vmem as hard resource requirement has been deleted"`. This message will appear on the stdout stream of the submit client application.

Reporting Verification Result

One of the following functions has to be called at the end of the `jsv_on_verify()` function after the job verification is done and just before `jsv_on_verify()` returns.

- `jsv_accept(message)`

A call of this function indicates that the job that is currently being verified should be accepted as it was initially provided. All job modifications that might have been applied before this function was called will be ignored. `message` parameter has to be a character sequence or an empty string. In the current implementation this string is ignored and it will appear only if logging for JSV is enabled.

- `jsv_correct(message)`

The job that is currently verified when this function is called will be accepted by the current JSV instance. Modifications that were previously applied to the job will be committed. The job will be either passed to the next JSV instance if there is one or it is passed to sge_qmaster so that it can be added to the masters data store when the function returned. `message` parameter has to be a character sequence or an empty string. In the current implementation this string is ignored and it will appear only if logging for JSV is enabled.

- `jsv_reject(message)` or `jsv_reject_wait(message)`

The currently verified job is is rejected. `message` parameter has to be a character sequence or an empty string. `message` will be passed as error message to the client application that tried to summit the job. Command line clients like `qsub` will print this `message` to notify the user that the submission has failed.

Accessing the Job Environment

The following function can be used to access the job environment that will be made available when the job starts. At the command line this environment is formed with the command line switches `-v` and `-V`. The function can only be used when `jsv_send_env()` was previously called in `jsv_on_start()`.

- `jsv_is_env(env_name)`

Returns `true` when an environment variable with the name `env_name` exists in the job currently being verified. In this case `jsv_get_env(env_name)` can be used to retrieve the value of that variable.

- `jsv_get_env(env_name)`

Returns the value of a variable named `env_name`. If the variable is not available an empty string will be returned. To distinguish non-existing variables and empty variables the function `jsv_is_env(env_name)` can be used.

- `jsv_add_env(env_name, env_value)` and `jsv_mod_env(env_name, env_value)`

These functions add or modify a environment variable named `env_name`. The value of the variable will be `env_value`. If `jsv_add_env()` is used on a variable that already exists then simply the value is overwritten, when `jsv_mod_env()` is used on a variable that does not already exist then it is silently added. `env_name` might be an empty string, in this case only the variable is set.

- `jsv_del_env(env_name, env_value)`

Removes `env_name` from the set of environment variables that will be exported to the job environment when the job is started.

Parameter Names of JSV Job Specifications

JSV functionality allows it to change various aspects of jobs that should be submitted to Univa Grid Engine systems. This can be done with predefined JSV script interface functions. Those function require valid parameter names and corresponding values. The table blow mentions all supported parameter names and describes them in more detail.

Parameter	Description
`a`	If a job has a specific start time and date at which it is eligible for execution (specified with `qsub -a` at the commandline) then the corresponding value is available in JSV as parameter with the name `a`. The value of this parameter has the following format: `[[CC]YY]MMDDhhmm[.ss]` where `CC` denotes the century, `YY` the year, `MM` the month, `DD` the day, `hh` the hour, `mm` the minutes and `ss` the seconds when the job could be started.
`ac`	The value for the ac parameter represents the job context of a job as it is specified at the command line with the command line switches `-ac`, `-dc` and `-sc`. The outcome of the evaluation of all three switches will be passed to JSV as the list parameter named `ac`. It is possible within JSV scripts to modify this list with the `jsv_sub_*_param()` functions.
`ar`	The `ar` parameter is available in JSV if a advance reservation number was specified during the submission of a job. At the command line this is done with the `-ar` switch. The value of `ar` can be changed in JSV as long as the new value is a valid advance reservation id.
`b`	If the parameter named `b` is available in JSV this shows that a job was submitted as binary job e.g with `-b` switch at the command line. The value in this case is `yes`. The absence of this parameter indicates that a non-binary job was submitted. Independent if the parameter is available or not it can be set or changed.

Parameter	Description
c_interval c_occasion	The command line switch -c of qsub can be used to define the occasions when a checkpointing job should be checkpointed. If a time interval is specified then this value will be available in JSV as a parameter with the name c_interval and when certain occasion are specified through characters then this letter is available through the parameter c_occasion. Is is possible to change both values in JSV. Note that a change of c_occasion will automatically override the current value of c_interval and vice versa.Valid values for c_occasion are the letters n, s, m and x where n disables checkpointing, s triggers a checkpoint when an execution daemon is shut down, m checkpoint at minimum CPU interval and x checkpoints when the job gets suspended.The time value for c_occasion has to be specified in the format hh:mm:ss where hh denotes hours, mm denotes minutes and ss the seconds of a time interval between two checkpointing events.
ckpt	The ckpt parameter is set for checkpointing jobs and contains the name of the checkpointing environment that can be defined at commandline with the -ckpt switch.
cwd	The value of the cwd parameter is if available the path to the working directory where the submit client was started. At the command line this will be set with the -cwd switch.
display	The value of display is used by xterm to contact the X server. At the command line the value for this parameters can be set in qsh and qrsh with the -display switch. The format of the display value has always to start with a hostname (e.g hostname:1). Local display names (e.g. :13) cannot be used in grid environments. Values set with the display variable in JSV will overwrite settings from the submission environment and environment variable values specified with -v command line option.
dl	The dl parameter is available if a deadline time was specified during the submission of a job. At the command line this can be done with the -dl switch. If available the value will have the same format as the a parameters that specifies the start time of a job.
e	The e parameter defines or redefines the path used for the standard error stream of a job. At the command line the value for this parameter can be defined with -e switch.
h	The value of the h parameter indicates that a job was submitted in user hold state (e.g. with qsub -h). In this case the parameter is available and it will be set to u. To change this the parameter can be set to n.
hold_jid	The hold_jid parameter contains job dependency information of a job. It is available when a job was submitted with -hold_jid command line switch. If available the list contains references in form of job ids, job names or job name patterns. Referenced jobs in this list have to be owned by the same user as the referring job.

Parameter	Description
hold_jid_ad	The hold_jid_ad parameter defines or redefines array job interdependencies. It is available when a job was submitted with -hold_jid_ad command line switch. If available the list contains references in form of job ids, job names or job name patterns. Referenced jobs in this list have to be owned by the same user as the referring job.
i	The i parameter defines or redefines the path used for the standard input stream of a job. At the command line the value for this parameter can be defined with -i switch.
j	Similar to the -j command line switch the j parameter defines or redefines if the standard error stream should be merged into the output stream of a job. In this case the parameter is available and set to y. To change this, the value can get set to n.
jc	Defines or redefines the job class that should be used to create the new job.
js	Defines or redefines the job share of a job relative to other jobs. If the corresponding -js parameter was not specified during submission of a job then the default job share is 0. In this case the parameter is not available in JSV. Nevertheless it can be changed.
l_hard l_soft	At the command line job resource requests are specified with the -l switch. This switch can be used multiple times also in combination with the switches -hard and -soft to express hard and soft resource requirements of a job. The sum of all hard and soft requests a job has will be available in JSV with the two parameters l_hard and l_soft. Note that if regular expressions or shortcut resource names were used in the command line then these expressions will also be passed to JSV. They will not be expanded. It is possible within JSV scripts to modify these resource list with the jsv_sub_*_param() functions.
m	The value of the m parameter defines or redefines when Univa Grid Engine sends mail to the job owner. Format is similar to the command line switch -m of the qsub command. n means that there is no mail sent and different letter combinations of the letters b, e, a and s can be used to define when mail is sent where b means that mail is sent at the beginning of a job, e at the end of a job and a when the job is aborted or rescheduled.
M	M is the list of mail addresses to which the Univa Grid Engine system sends job related mails. It is possible within JSV scripts to modify these resource list with the jsv_sub_*_param() functions.
masterq	masterq parameter defines candidate queues that might become the master queue if the submitted job is a parallel job. At the command line this is specified with the -masterq commandline switch. In JSV the list can be accessed with the jsv_sub_*_param() script functions.

Parameter	Description
notify	Jobs where the notify parameter is available in JSV and where it is set to y will receive a notify signal immediately before a suspend or kill signal will be delivered. If -notify was not used during submission of a job then the notify parameter will not be available.
now	Not available in JSV.
N	The value of N is the job name of the job to be submitted.
o	The o parameter defines or redefines the path used for the standard output stream of a job. At the command line the value for this parameter can be defined with -o switch.
p	The p parameter defines or redefines job priority relative to other jobs. It is only available if the value is not equal 0. Allowed values for this parameter are integer values in the range between -1023 and 1024.
pe_name pe_min pe_max	When parallel jobs are submitted with qsub, the -pe command line switch has to be specified to define which parallel environment should be used and also the needed slots can be defined. The parameters pe_name, pe_min and pe_max show parts of that specification. pe_name is the name of the parallel environment. pe_min and pe_max specify the biggest and smallest slot number used in the slot specification. If multiple pe ranges are specified pe_n will contain the number of ranges, pe_min_0 the minimum of the first range, pe_max_0 the maximum of the first range, pe_min_1 the minimum of the second range, ...
pty	This parameter is only available in Univa Grid Engine 8.0.1 and above (see UNIVA_EXTENSIONS pseudo parameter below). The -pty switch of qrsh and qsh enforces the submitted job to be started in a pseudo terminal. This information will be exported to client and server JSV scripts with the parameter named pty. If the command line switch is omitted then then this parameters has the value u which means unset. Client application and executed job will use the default behavior. y means that the use of a pseudo terminal is enforced and n that no pseudo terminal will be used. This parameters can be changed in JSV scripts. This change will influence the client application and the executed job as if the corresponding command line switch would have been used directly.
P	Variable that holds the project name to which a job is assigned. A change of this value will overwrite the value specified with the -P command line parameter.
q_hard q_soft	The -q switch at the command application can be combined with the -hard and -soft switches. As a result the user specifies lists of hard and soft cluster queue, queue domain and queue instance requests. Within JSV those lists are available via the parameters q_hard and q_soft. Both of them can be changed using the jsv_sub_*_param() script functions.

Parameter	Description
R	If the R parameter is available and set to y when a reservation will be done for the corresponding job. The request for reservation can be undone by JSV when the parameter is set to n.
r	Jobs can be marked to be rerunnable with the -r command line switch. If this has been done then the | r parameter is available and set to y. To overwrite this the value can be changed to n.
shell	The parameter shell is defined and set to y if a command shell should be used to start the job. To disable this JSV value has to be changed to n.
sync	This parameter is only available in Univa Grid Engine 8.0.1 and above (see UNIVA_EXTENSIONS pseudo parameter below). When a command line application is used with the -sync command-line switch then within client and server JSV the parameters with the name sync will be available and it will be set to y. The sync parameter is a read-only parameter in JSV. This means that it is not possible to influence the behavior of the command line client by modifying this parameter in JSV.
S	The S parameters specifies the interpreting shell for the job.
t_min t_max t_setp	The -t parameter of qsub submits an array job. The task range specification is available in JSV via three parameters: t_min, t_max and t_step. All three values can be changed.
terse	This parameter is only available in Univa Grid Engine 8.0.1 and above (see UNIVA_EXTENSIONS pseudo parameter below). When a command line application is used with the terse-switch then the parameter named terse will be available in client and server JSV scripts and it will be set to y. If this parameters is set to n then the submit client will print the regular "Your job ..." message instead of the job ID. The parameter value can be changed within JSV scripts.
v	There is no v parameter in JSV. If information concerning the resulting job environment is needed in JSV then this has to be requested explicitly. When using the JSV script interface, this can be done with a call of jsv_send_env() in jsv_on_start(). After that the jsv_*_env() functions can be used to access the job environment.
V	This parameter is only available in Univa Grid Engine 8.0.1 and above (see UNIVA_EXTENSIONS pseudo parameter below). The V parameter will be available in client and server JSV scripts and it will have the value y when the -V command line switch was used during the submission of a job. This indicates that the full set of environment variables that were set in the submission environment can be accessed from JSV. If this parameter is not available or when it is set to n then only a subset of the user environment can be accessed in JSV scripts. Only those environment variables will be available that were passed with the -v command line parameter.

Parameter	Description
wd	See cwd

Table 85: JSV Job Parameter Names

Additionally to the job parameters JSV provides a set of pseudo parameters

Parameter	Description
CLIENT	The value of the CLIENT is either qmaster in case of server JSV or for client JSV's the name of the submit client that tries to submit the job. Valid client names are qsub, qrsh, qsh, qlogin or qmon. In case of DRMAA clients the string drmaa is used. This value is read-only. It cannot be changed by JSV.
CMDARG{i}	Command line arguments of the job script will be available within JSV via multiple CMDARG{i} parameters where {i} is replaced with by the number of the position where the argument should appear. {i} is a number in the range starting from 0 to CMDARGS - 1. This means that the first argument will be available through the parameter CMDARG0.
CMDARGS	The value is a integer number that representing the number of command line arguments that should be passed to the job when it is started.
CMDNAME	In case of binary submission the CMDNAME contains the command name of the binary to be executed, for non-binary jobs the full path to the job script is specified. Within Univa Grid Engine systems it is allowed to modify CMDNAME in client and server JSV scripts. CMDNAME might be set to a script name or to a command name in case of binary submission. If script submission should be changed to binary submission or binary submission should be changed to script submission then the b parameter has to be changed in JSV before the CMDNAME parameter is changed. In case of script submission the script has to be accessible on the master machine if the CMDNAME parameter is changes in a server JSV. Submission parameters that are embedded in the new script file will be ignored.
CONTEXT	The CONTEXT might have two values. client or master depending on which client host the JSV is currently executed. It is not possible to change this value.
GROUP	The value of GROUP is the primary group name of the user who submitted the job. Cannot be changed by JSV.
SUBMIT_HOST	This parameter is only available in Univa Grid Engine 8.0.1 and above (see UNIVA_EXTENSIONS pseudo parameter below). Within server JSV's the read-only parameter SUBMIT_HOST is available. This parameter contains the host-name where the submit application is executed.

Parameter	Description
JOB_ID	This variable is not available when CONTEXT is client (client JSV). In case of server JSV the value of JOB_ID is the job number the job would get when it is accepted by the Univa Grid Engine system. This value cannot be changed by JSV.
UNIVA_EXTENSIONS	The JSV parameter named UNIVA_EXTENSIONS is available in Univa Grid Engine 8.0.1 and above. This read-only parameter can be used in client and server JSV scripts to detect if a certain set of JSV parameters can be accessed that are only available in the Univa version of Grid Engine. If this parameter is not available or when it is set to n then these extensions to JSV are missing (Open Source version of Grid Engine). In this case it is not possible to access following parameters: pty, sync, terse, V and SUBMIT_HOST.
USER	The value of USER is the Unix user name of the user who submitted the job. Cannot be changed by JSV.
VERSION	Shows the VERSION of the implementation of the JSV communication protocol. VERSION is always available in JSV and it is not possible to change the value. The format of the value is {major}.{minor}. Since the first implementation of JSV the communication protocol has not been changed so that the current VERSION is still 1.0

Table 86: JSV Pseudo Parameter Names

Using JSVs for Integrating Univa Grid Engine With Other Facilities

Script bases JSVs are the best compromise concerning performance and flexibility to align jobs according to the needs that predominate a cluster. Other facilities that should be integrated with Univa Grid Engine might have different requirements. Such facilities might require that:

- a special programming language is used.
- certain tasks should be achieved that cannot be done easily within a script language.
- performance has to be optimized so that cluster throughput can be increased.

To be able to do so Univa Grid Engine provides information about the communication protocol that is used between acting components so that administrators are able to write JSVs in any programming language. Contact us to receive more detailed information.

The JSV protocol has meanwhile been implemented for the Java programming language. JAR files as well as documentation are part of the distribution. Find it in the directories $SGE_ROOT/util/resources/jsv, $SGE_ROOT/lib.

The Java implementation has been used for the Hadoop integration which is explained in a different chapter.

3.8.8 Enabling and Disabling Core Binding

The core binding feature can be enabled and disabled on host level. In a default Univa Grid Engine installation, it is turned on for Linux hosts, while on Solaris architectures it must be enabled by the administrator. The reason for this is that the functionality differs on these two supported architectures. On Linux a bitmask is set for a process, which tells the operating system scheduler **not** to schedule the process to specific cores. The net result is a more streamlined processed. The scheduler does not prevent other processes to be scheduled to the specific cores (nevertheless it avoids it). On a Solaris processor sets are used. They require **root privileges** and exclude other processes (even OS processes) to run. Hence it would be possible by the user to occupy cores, even when the application is granted just one slot. In order to avoid this, the administrator must ensure the number of cores are aligned with the number of granted slots. This can be done with advanced JSV scripts.

To turn this feature on, add `ENABLE_BINDING=true` to the `execd_params` on the specific execution host. The feature is explicitly disabled with `ENABLE_BINDING=false`.

Example: Enabling Core Binding on Host `host1`

```
> qconf -mconf host1
mailer                      /bin/mail
...
execd_parameter             ENABLE_BINDING=true
```

3.9 Ensuring High Availability

For an introduction to the shadow master concept see also Introduction Guide -> Concepts and Components -> SGE_SHADOWD and the Shadow Master Hosts.

With one Univa Grid Engine installation or multiple instances, sge_shadowd can monitor sge_qmaster availability and in case of sge_qmaster outages, start a new sge_qmaster on a shadow host.

The shadow master functionality uses the following algorithm: * during regular operation sge_qmaster writes a heartbeat file in regular intervals (written every 30 seconds to file `<qmaster spool dir>/heartbeat`)

- all sge_shadowd instances monitor the heartbeat file
- if a sge_shadowd detects that the heartbeat file has not changed for a certain time (see Tuning the sge_shadowd, it tries to take over the qmaster role, according to the following algorithm:
  - avoid multiple instances of sge_shadowd takeover (via a lock file)
  - check if the old qmaster is still down
  - startup sge_qmaster

3.9.1 Prerequisites

Implementing high availability via sge_shadowd requires a specific setup regarding sge_qmaster spool directory and spooling method:

- all shadow hosts must be administrative hosts
- the sge_qmaster spool directory must be shared amongst the master host and all the shadow hosts
- for qmaster spooling, the following options can be used:
  - classic spooling on a shared file system
  - berkeleydb spooling on a shared file system providing locking capabilities, e.g. NFS4 or Lustre. The master host and all shadow hosts must have the same architecture (Univa Grid Engine architecture string)

See also Installation Guide -> Selecting a File System for Persistency Spooling of Status Data - Selecting a File System for Persistency Spooling of Status Data for selecting the spooling method and file system.

3.9.2 Installation

For the installation of shadow hosts see Installation Guide -> Shadow master host installation - Shadow master host installation.

3.9.3 Testing sge_shadowd Takeover

After doing the shadow host installation on one or multiple shadow hosts, make sure the shadowd takeover actually works.

To test shadowd takeover, simulate the outage of the sge_qmaster or of the master host by either:

- unplugging the network interface of the master host
- suspending or terminating the sge_qmaster process (do not gracefully shutdown sge_qmaster - in this case, sge_shadowd will not take over)

Monitor Univa Grid Engine functionality by calling qstat in regular intervals - qstat will fail until one of the shadow hosts has taken over control.

When the shadowd mechanism has started up a shadow master, check `$SGE_ROOT/$SGE_CELL/common/act_qmaster` - it will contain the name of the new master host.

Monitor with qhost if all execution hosts start using (register with) the sge_qmaster on the shadow host.

3.9.4 Migrating the Master Host Back After a Takeover

It may be necessary to manually migrate sge_qmaster to a different host, e.g.

- when some maintenance on the master host is done, migrate sge_qmaster to one of the shadow hosts
- after a shadow host takes over, migrate back sge_qmaster to the original master host

On the target sge_qmaster host as user root call `$SGE_ROOT/$SGE_CELL/common/sgemaster -migrate`

This command

- shuts down the running sge_qmaster
- starts up a sge_qmaster on the local host.

3.9.5 Tuning the sge_shadowd

Timing behavior of sge_shadowd can be configured via 3 environment variables:

- SGE_CHECK_INTERVAL: Controls the interval in which sge_shadowd checks the heartbeat file. The default is 60 seconds. sge_qmaster writes the heartbeat file every 30 seconds.
- SGE_GET_ACTIVE_INTERVAL: When the heartbeat file has not changed this number of seconds, sge_shadowd will try to take over. Default is 240 seconds.
- SGE_DELAY_TIME: If a sge_shadowd tried to take over, but detected that another sge_shadowd already started the take over procedure, it will wait for SGE_DELAY_TIME seconds until it takes up checking the heartbeat file again. Default is 600 seconds.

See also the man page sge_shadowd.8.

Be careful tuning these parameters. Setting the values too small may result in sge_shadow taking over in situations where the sge_qmaster has short outages, e.g. short network outages, or delays in propagating the contents of the heartbeat file from the master host to shadow hosts due to high load on the NFS server.

Recommendation:

- Start with the default values. This will result in a shadowd take over to happen within some 6 minutes.
- Reducing the SGE_GET_ACTIVE_INTERVAL is safe, e.g. setting it to 10 seconds can reduce the takeover time by some 50 seconds.

- Most benefit can come from tuning the SGE_GET_ACTIVE_INTERVAL parameter. Setting the value too low can result in sge_shadowd trying to take over when short outages occur, e.g. due to short network or NFS server outages / overload. Setting it for example to 60, and setting SGE_GET_ACTIVE_INTERVAL to 10 seconds can result in a shadow host takeover time of some 70 seconds.
- Tuning the SGE_DELAY_TIME should usually not be necessary - it would be used to reduce the time interval for a second shadow host take over if the first shadow host fails to take over. Be careful tuning this parameter. It should never be lower than the time required for starting sge_qmaster in the cluster. Sge_qmaster startup time depends on cluster size and the number of jobs being registered in the cluster. In a big cluster with thousands of jobs being registered, the sge_qmaster startup time can be in the magnitude of minutes.

3.9.6 Troubleshooting

How do I know which host is currently running sge_qmaster?

The name of the host running sge_qmaster can be found in the `$SGE_ROOT/$SGE_CELL/common/act_qmaster` file.

Where do I find run time information of running shadow daemons?

Every sge_shadowd writes run time information into its own messages file, which can be found at `<qmaster spool dir>/messages_shadowd_<hostname>`. It contains information about the running sge_shadowd, e.g. its version, as well as monitoring information and reports about shadow host activity, e.g.

```
05/02/2011 11:19:16|  main|halape|I|starting up UGE 8.0.0 beta (lx-x86)
05/02/2011 11:40:18|  main|halape|E|commlib error: got select error (No route to host)
05/02/2011 11:40:18|  main|halape|W|starting program: /scratch/joga/clusters/shadow/
                                                      bin/lx-x86/sge\_qmaster
```

Startup of sge_qmaster on a shadow host failed. Where do I find information for analyzing the problem?

The file `<qmaster spool dir>/messages_qmaster.<hostname>` contains the time when sge_shadowd on started a sge_qmaster, as well as sge_qmaster output to stdout and stderr at startup time.

3.10 Utilizing Calendar Schedules

Calendar objects within Univa Grid Engine are used to define time periods where certain cluster resources are disabled, enabled, suspended or unsuspended. Time periods can be be defined on a time of day, day of week or day of year basis.

Defined calendar objects can be attached to cluster queues or parts of cluster queues so that it automatically changes its state on behalf of that attached calendar.

Users submitting jobs can request queues with a certain calendar attached.

3.10.1 Commands to Configure Calendars

To configure a calendar, the qconf command can be used which provides a number of calendar related options:

- `qconf -acl calendar_name`

The 'add calendar' options adds a new calendar configuration named calendar_name to the cluster. When this command is triggered, an editor with a template calendar configuration will appear.

- `qconf -Acl filename`

This command adds a calendar specified in filename to the Univa Grid Engine system.

- `qconf -dcal calendar_name`

The 'delete calendar' option deletes the specified calendars.

- `qconf -mcal calendar_name`

The 'modify calendar' option shows an editor with an existing calendar configuration of the calendar named calendar_name.

- `qconf -scal calendar_name`

The 'show calendar' option displays the configuration of the calendar calendar_name.

- `qconf -sscal`

The 'show calendar' list options shows all configured calendars of a Univa Grid Engine system

3.10.2 Calendars Configuration Attributes

A calendar configuration allows the following configuration attributes:

Attribute	Value specification
`calendar_name`	The name of the calendar to be used when attaching it to queues or when administering the calendar definition.
`year`	The status definition on a day of the year basis. This field generally will specify the days on which a queue, to which the calendar is attached, will change according to a set state. The syntax of the year filed is defines as follows: `NONE`

Attribute	Value specification

	`year_day_range_list = daytime_range_list [= state]`
	`year_day_range_list [= daytime_range_list] = state`
	`state`
	* `NONE` means no definition is made on the year basis.
	* If a definition is made on a yearly basis, at least one of the `year_day_range_list`, `daytime_range_list` and state have to be present.
	* switching the queue to 'off' by disabling it assumes the state is omitted.
	* the queue is enabled for days neither referenced implicitly by omitting the `year_day_range_list` nor explicitly.
	and the syntactical components are defined as follows:
	`year_day_range_list := yearday-yearday | yearday, ...`
	`daytime_range_list := hour[:minute][:second]-`
	`hour[:minute][:second], ...`
	`state := on | off | suspended`
	`year_day := month_day.month.year`
	`month_day := 1 | 2 | ... | 31`
	`month := jan | feb | ... | dec | 1 | 2 | ... | 12`
	`year := 1970 | 1971 | ... | 2037`

`week`	The status definition on a day of the week basis. This field generally will specify the days of a week and at the times at which a queue, to which the calendar is attached, will change to a certain state. The syntax of the week field is defined as follows:
	`NONE`
	`week_day_range_list[=daytime_range_list][=state]`
	`[week_day_range_list=]daytime_range_list[=state]`
	`[week_day_range_list=][daytime_range_list=]state}`
	`...`
	Where
	* `NONE` means, no definition is made on the week basis
	* if a definition is made on the week basis, at least one of `week_day_range_list`, `daytime_range_list` or state always have to be present.
	* every day in the week is assumed if week_day_range_list is omitted.

Attribute	Value specification
	* syntax and semantics of daytime_range_list and state are identical to the definition given for the year field above. * the queue is assumed to be enabled for days neither referenced implicitly by imitating the `week_day_range_list` nor explicitly and where week_day_range_list is defined as`week_day_range_list := week_day-week_day | week_day, week_day := mon | tue | wed | thu | fri | sat | sun` with week_day_range_list the week_day‘ identifiers must be different.

Table 87: Calendar configuration attributes

Note that successive entries to the year or week fields (separated by blanks) are combined in compliance with the following rules:

- off-areas are overridden by overlapping on- and suspend-areas. Suspend-areas are overridden by on-areas. Hence an entry of the form `week 12-18 tue=13-17=on` means that queues referencing the corresponding calendar are disabled the entire week from 12.00-18.00 with the exception of Tuesday between 13.00-17.00 where the queues are available.
- area overriding occurs only within a year or week basis. If a year entry exists for a day then only the year calendar is taken into account and no area overriding is done with a possible conflicting week area.
- The second time specification in a `daytime_range_list` may be before the first one and treated as expected. An entry like `year 12.3.2011=12-11=off` causes the queue(s) to be disabled 12.3.2011 from 00:00:00-10:59:59 and 12:00:00-23:59:59.

3.10.3 Examples to Illustrate the use of Calendars

```
calendar_name  night
year
1.1.1999,6.1.1999,28.3.1999,30.3.1999-31.3.1999,18.5.1999-19.5.1999,
3.10.1999,25.12.1999,26.12.1999=on
week           mon-fri=6-20
```

- The calendar configuration above defines a night, weekend and public holiday calendar
- On public holidays, night queues are explicitly enabled.
- On working days, queues are disabled between 6.00 and 20.00.
- Saturdays and Sundays are implicitly handled as enabled times.

  calendar_name day year
  1.1.1999,6.1.1999,28.3.1999,30.3.1999-31.3.1999,18.5.1999-19.5.1999, 3.10.1999,25.12.1999,26.12.1999
  week mon-fri=20-6 sat-sun

- On public holidays day-queues are disabled.
- On working days such queues are closed during the night between 20.00 and 6.00, i.e. the queues are closed on Monday from 0.00 to 6.00 and on Friday from 20.00 to 24.00. On Saturdays and Sundays the queues are disabled.

  calendar_name night_s year
  1.1.1999,6.1.1999,28.3.1999,30.3.1999-31.3.1999,18.5.1999-19.5.1999, 3.10.1999,25.12.1999,26.12.1999=on week mon-fri=6-20=suspended
- night_s is a night, weekend and public holiday calendar with suspension.
- Essentially the same scenario as the first example in this section but queues are suspended instead of switched off.

  calendar_name weekend_s year NONE week sat-sun=suspended
- Weekend calendar with suspension, ignoring public holidays.
- Settings are only done on the week basis, no settings on the year basis.

3.11 Setting Up Nodes for Exclusive Use

Administrators can set up Univa Grid Engine in a way so that users can request hosts for exclusive use independent of how many processors or cores are provided. This is done independently if the host is used for single core batch jobs, bigger parallel jobs, or something different.

Exclusive host usage might help:

- execute jobs independently that would otherwise interfere with each other jobs or with system resources that can be only used exclusively.
- set up security terms required for certain jobs

To enable hosts of a Univa Grid Engine cluster to be used for exclusive use, the administrator has to:

- Add an exclusive boolean consumable to the complex definition that specifies as relop the `EXCL` keyword and that is requestable. The `qconf -mc` command can be used to do so .

  #name shortcut type relop requestable consumable default urgency exclusive excl BOOL EXCL YES YES 0 1000
- Attach the consumable to hosts that should be used exclusively. This is done by using the qconf -me host_name command. exclusive=true has to be added to the `complex_values` of the corresponding host.

Users who want to request a host exclusively have to

- specify the consumable during job submission. This is done with `-l exclusive=true` parameters with the command line applications

  qsub -l exclusive=true

3.12 Deviating from a Standard Installation

3.12.1 Utilizing Cells

Univa Grid Engine can be set up so that the resources participating in a single cluster or multiple individual clusters sharing the same set of files (binaries, libraries, É) contained in the `$SGE_ROOT` directory can be set up.

If multiple clusters are set up then these are uniquely identified by the `$SGE_CELL` environment variable set during cluster installation which contains a unique cell name that remains valid until the cluster is uninstalled. Recommended cell name for the first installed cluster is `default`.

After installing a Univa Grid Engine cell, the configuration files spools files of that cell. These can be located in `$SGE_ROOT/$SGE_CELL`.

Note that at the moment, cells are loosely coupled so that each cell has the full set of daemons and other components that act independently from the daemons and components participating in other cells. So there is no automatic means to balance load between those clusters.

3.12.2 Using Path Aliasing

The Univa Grid Engine path aliasing facility provides administrators and users with the means to reflect in-homogeneous file system structures in distributed environments. One example for this are home directories that are mounted under different paths on different hosts.

Consider a user home directory that is exported via NFS or SMB. This directory might be mounted via automounter to /home/username on some Linux hosts and to /Users/username on host with Mac OS X as operating system. On Solaris host /home/username might be a link to /tmp_mnt/home/username where the directory was mounted by the automounter.

If a user submits a job using the -cwd switch somewhere within the home directory then the job needs the current working directory to be executed successfully. If a job's execution host is one where the home directory is mounted differently then the system will not be able to locate the directory on the execution environment.

To solve this problem Univa Grid Engine provides the possibility for administrators to define a global path aliasing file in `$SGE_ROOT/$SGE_CELL/common/sge_aliases`. Users can also define a path aliasing file in the directory `$HOME/.sge_aliases`.

The Format of the file is as follows:

- Empty lines and lines beginning with a hash character (#) will be skipped.
- Other lines must contain four strings separated by space or tab characters
- First string specifies a source path the instant a host is submitted
- Third string defines an execution host and the fourth string a destination path
- Submit hostname and execution hostname might be replaced by an asterisk character (*) that matched any hostname.

If the -cwd flag to qsub is specified then the path aliasing mechanism is activated and the defined files are processed as follows:

- The global path aliasing file is read.
- The user path aliasing file is read if present and it is appended to the global file.
- Lines not skipped will be processed from top to bottom.
- All lines are selected where the hostname matches the submit hostname. The submit client is executed where the source path forms the initial part of the current working directory or one of the source path replacements that were previously selected.
- All selected entries are passed along with the job to the execution host.
- The leading part of the current working directory on the execution host are replaced by the source path replacement where execution host string matches. The current working directory is changed further when there are entries where the host string and the initial part of the modified working directory matches.

Here is an example for a path aliasing file that replaces the occurrence of /tmp_mnt/ by /.

```
# Path Aliasing File
# src-path   sub-host     exec-host   replacement
/tmp_mnt     *            *           /
```

3.12.3 Host-name Resolving and Host Aliasing

For host-name resolving Univa Grid Engine is using the standard UNIX directory services like DNS, NIS, and /etc/hosts depending on the operating system configuration. The resolved names provided by such services are also cached within the communication library. The host-name cache is renewed from time to time. It is possible to change the re-resolving timeouts with the following sge_qmaster configuration parameters (see also man sge_conf(5)):

- DISABLE_NAME_SERVICE_LOOKUP_CACHE
- NAME_SERVICE_LOOKUP_CACHE_ENTRY_LIFE_TIME
- NAME_SERVICE_LOOKUP_CACHE_ENTRY_UPDATE_TIME
- NAME_SERVICE_LOOKUP_CACHE_ENTRY_RERESOLVE_TIME

In rare cases these standard services cannot be setup cleanly and Univa Grid Engine communication daemons running on different hosts are unable to automatically determine a unique hostname for one or all hosts which can be used on all hosts. When packages from the qmaster arrive at execution daemons (or vice versa) then they might get rejected when they come from a different IP address/host-name than the daemons are expecting. In such situations a host aliases file can be used to provide the communication daemons with a private and consistent hostname resolution database.

Hence Univa Grid Engine allows to configure a **host_aliases** file. The file has to be located in the `$SGE_ROOT/$SGE_CELL/common/` directory of the installation. This file does a mapping from all allowed host-names to some unique host-name, which is known by the daemons through the Univa Grid Engine configuration.

Changes to the host_aliases file are not immediately active for components that are already running. If the changes result e.g. in a different host-name for an execution daemon the daemon must be restarted. At startup of sge_execd or sge_qmaster the database configuration is verified and adjusted. This is also the case if the resulting host-name of the UNIX directory services have changed. If the name of a sge_qmaster or sge_execd host has changed at the UNIX directory services during runtime the running components should be restarted to trigger also a verification of the database. Without a restart of such daemons the change will be effective once the cached value of the resolved host-name is renewed and it might result in unexpected behavior if previously used host-names are not resolveable or not unique anymore.

Adding new entries without restarting of the sge_qmaster daemon is possible if the resulting host-names are not influencing the cluster configuration. The depending configurations are host configurations, admin host names, execd host names and submit host names. The sge_qmaster daemon will re-read the host_aliases file during runtime from time to time and add some information into the messages logging file. If it would be necessary to restart the sge_qmaster daemon this will also result in a logging in the sge_qmaster messages file.

Note

If an already used host-name should be changed either in the directory services or in the host_aliases file without restarting of the sge_qmaster the affected host might be removed from Univa Grid Engine first. Once all references to the old host-name are removed and all daemons running on that host have been shut down, the host-name can be changed. Once the host-name has been changed the previous name could still be cached in the communication library or in system services like `named`. Please make sure that the running sge_qmaster resolves the host-name correctly before adding the renamed host again. This verification could be done with **`gethostbyname -all_rr <hostname>`**.

Note

After creating or changing the host_aliases file some daemons might have to be restarted. This can be done by e.g. qconf -ke, qconf -km, manually shutting down the shadowd, and finally run inst_sge -start-all to restart all daemons.

Note

Making changes of the behavior of the standard UNIX directory services might also make it necessary to restart affected Univa Grid Engine daemons.

Note

If it is required to restart both sge_qmaster and some sge_execd daemons the sge_qmaster must have been shutdown before restarting all other Univa Grid Engine components.

In order to test the resolution **`gethostbyname -aname <alias>`** can be called on command line. When testing be sure that `<alias>` itself can be resolved. When the alias is not known by the UNIX directory service than it will not work correctly and gethostbyname will report an error. In that case use system commands like **ping** to check if the host-name can be resolved by the system itself.

In order to figure out the resulting name of a host at sge_qmaster **`gethostbyname -all_rr <hostname>`** could be used (see also man hostnameutils(1) man page). It will additionaly show

the name resolving of the `<hostname>` name on the current sge_qmaster host. In order to use this option the sge_qmaster daemon must be reachable.

Format of the host_aliases file

Each line of the **host_aliases** file contains a space separated list of host aliases. The first entry is always the unique host-name.

Example:

```
SLES11SP1 SLES11SP1_interface2
```

In this example the host has two network interfaces. With this host_aliases line Univa Grid Engine components will choose the "SLES11SP1" interface for binding a port or for opening a connection.

Note
All used host-names must be resolveable via some reachable directory service otherwise the entry is ignored.

Example:

```
hostfoo hostfoo.domain somehost somehost.domain
```

In this example the resolveable hosts "hostfoo.domain", "somehost" and "somehost.domain" would be all result in the unique host-name "hostfoo".

3.13 Using CUDA Load Sensor

The CUDA load sensor enables the Univa Grid Engine installation GPU device awareness for extended execution host status tracking and for CUDA aware scheduling purposes.

The following metrics will be reported by this additional load sensor:

- Total installed GPU memory
- Total memory allocated
- Total free memory
- EC
- C Errors Total
- GPU temperature
- Power usage (in milliwatts)

These values will be reported by the load sensor and allow the end-user of Univa Grid Engine to specify their requirements as part of the resource request list in `qsub`. Of particular mention is the fact that when the hardware does not support ECC memory, so *eccEnabled* reports 0 and the ECC error counts are unavailable.

Beginning with Univa Grid Engine 8.0.1 a sample CUDA loadsensor (written in C) is provided as source code as well as a pre-compiled binary (for `lx-amd64`). The pre-compiled binary was build against `CUDAtools` and `CUDAtoolkit` version 4.0.17. Both files are located in the `$SGE_ROOT/util/resources/loadsensors/` directory, the pre-compiled binary in the `lx-amd64` directory below.

3.13.1 Building the CUDA load sensor

The CUDA load sensor source code (non-open source) is available at:

```
`$SGE_ROOT/util/resources/loadsensors/cuda_loadsensor.c`
```

The file should be moved to an appropriate directory where the load sensor binary is going to be built.

In order to build the load sensor following 3 packages must be available (they can be downloaded from NVIDIA):

- cudatools (tested with 4.0.17)
- cudatoolkit (tested with 4.0.17)
- the NVIDIA driver itself

In order to build the load sensor a `Makefile` similar to the one below can be used. The `CUDA_TOOLKIT_HOME, CUDA_TOOLS_HOME,` and `NVIDIA_DRIVER_HOME` variables must be adapted according to the download paths of the NVIDIA packages. The file has to be saved as Makefile in the directory of the `cuda_load_sensor.c` file. With the `make` command the load sensor can be built.

make

Example Makefile for building the CUDA load sensor

```
CUDA_TOOLKIT_HOME = /tools/PKG/cudatoolkit_4.0.17
CUDA_TOOLS_HOME = /tools/PKG/cudatools_4.0.17
NVIDIA_DRIVER_HOME = /tools/PKG/NVIDIA-driver-270.41.19

CUDA_CFLAGS=-Wall -g -Wc++-compat -Werror
CUDA_INCLUDE=-I$(CUDA_TOOLKIT_HOME)/include -I$(CUDA_TOOLS_HOME)/NVML
CUDA_LDFLAGS=-L$(CUDA_TOOLKIT_HOME)/lib64 -L$(NVIDIA_DRIVER_HOME)
CUDA_LIBS=-lnvidia-ml

CUDA_SRC=cuda_load_sensor.c
```

```
CUDA_OBJS=$(CUDA_SRC:.c=.o)

<nowiki>#</nowiki> global rules: all, clean, depend

all: cuda_load_sensor

clean:
   $(RM) $(CUDA_OBJS) cuda_load_sensor


<nowiki>#</nowiki> rules to build the CUDA load sensor

cuda_load_sensor: $(CUDA_OBJS)
   $(CC) -o cuda_load_sensor $(CUDA_OBJS) $(CUDA_LDFLAGS) $(CUDA_LIBS)

<nowiki>#</nowiki> rules to build object codes

cuda_load_sensor.o: cuda_load_sensor.c
   $(CC) $(CUDA_CFLAGS) $(CUDA_INCLUDE) -c cuda_load_sensor.c
```

3.13.2 Installing the load sensor

Add "cuda" complex value

A new `cuda` complex is needed in order to let Univa Grid Engine manage access to the GPU devices.

With `qconf -mc` the following value has to be added:

```
cuda      cuda       INT         <=    YES         JOB        0        1000
```

This defines a new complex value as a per job consumable, i.e. one job always consumes at most one instance of the complex (if requested and granted from the scheduler) whether it is a sequential or a parallel job (using more than one slot).

In order to make the values of the cuda load sensor usable within Univa Grid Engine they have to be added to the complex configuration as well. This can be done with the `install_cuda_complexes.sh` script, which is located in `$SGE_ROOT/util/resources/loadsensors`. The complexes are added with the `--install` argument:

```
./install_cuda_complexes.sh --install
```

and the complexes can be removed with the `--uninstall` argument.

```
./install_cuda_complexes.sh --uninstall
```

Assign "cuda" complex value to CUDA-enabled nodes

For all nodes (exchange their names with `hostname`) which have one CUDA GPU card installed the host complex `cuda` needs to initialized.

```
qconf -aattr exechost complex_values cuda=1 <hostname>
```

To remove the "cuda" complex value, following step must be performed:

```
qconf -dattr exechost complex_values cuda=1
```

Install the load_sensor

Use `qconf -mconf <hostname>` to add the load sensor (substitute `PATHTO` with the real path to the load sensor) to all hosts which have a CUDA load sensor installed:

```
load_sensor                  /PATHTO/cuda_load_sensor
```

Installation Test

Use the following command to test for the load sensor:

```
qconf -se <hostname>
```

Metrics reported by the CUDA load sensor should be reported in load_values.

Note
The load sensor will take at least one load report interval before it starts reporting values.

3.13.3 Using the CUDA load sensor

In order to submit a CUDA enabled application to hosts with GPU devices just the `cuda` complex must be requested during submission time.

```
qsub -l cuda=1 ... <your_application>
```

Then the Univa Grid Engine scheduler dispatches the job only to a host with an unused GPU device. If all GPU devices are in use by other Univa Grid Engine jobs, then the job remains in the pending job list (job state `qw`) waiting for a free device.

The CUDA load sensor itself reports numerous status values of the device. They can be used either for tracking the status or also for scheduling purposes. An example output (generated by an `qstat -F` statement) looks like following:

```
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@hostname                 BIPC  0/0/10         0.00     lx-amd64
   hl:cuda.verstr=270.41.06
   hl:cuda.0.name=GeForce 8400 GS
   hl:cuda.0.totalMem=511.312M
   hl:cuda.0.freeMem=500.480M
   hl:cuda.0.usedMem=10.832M
```

```
hl:cuda.0.eccEnabled=0
hl:cuda.0.temperature=44.000000
hl:cuda.1.name=GeForce 8400 GS
hl:cuda.1.totalMem=511.312M
hl:cuda.1.freeMem=406.066M
hl:cuda.1.usedMem=20.274M
hl:cuda.1.eccEnabled=0
hl:cuda.1.temperature=43.000000
hl:cuda.devices=2
```

3.14 Support of Intel® Xeon Phi™ Co-Processors

This section describes the enhancements made in Univa Grid Engine 8.1.3 for supporting Intel Xeon Phi enhanced compute clusters. The first sub-section demonstrates installation, configuration and usage of the Intel Xeon Phi load sensor. The second sub-section discusses the new RSMAP topology mask functionality designed to exploit NUMA architectures for a better application performance when multiple Intel Xeon Phi cards are installed. To install and configure Univa Grid Engine (beginning with 8.4.4) support for Intel Xeon Phi x200 (Knights Landing) Processors the following steps are not necessary. Please proceed with 'Configure and Install Intel Xeon Phi x200 (Knights Landing) Processors support'.

3.14.1 The Intel Xeon Phi Load Sensor

Univa Grid Engine supports managing arbitrary resources by resource surrogates (the so called complexes), which are handled by the scheduler. One source of the current state of such resources can be the load sensors. Load sensor are started on execution hosts by the Univa Grid Engine execution daemon. They follow a simple protocol in order to communicate new load values to the execution daemon, which in turn forwards the values to the Univa Grid Engine master process.

The Intel Xeon Phi load sensor is distributed in two forms, a pre-compiled `lx-amd64` binary (RHEL 6) and in raw C code form. The C code allows you to build the load sensor in your environment which might be different, for example, when driver and Intel SDK versions change. For both versions you need the Intel Xeon Phi drivers as well as the Intel MicAccessSDK installed. Special attention should be taken that the `libMicAccessSDK.so` file is in the `LD_LIBRARY_PATH`, when starting the load sensor (a wrapper script can be written to ensure this).

The Intel Xeon Phi binary load sensor is located in

`$SGE_ROOT/util/resource/loadsensors/phi/lx-amd64`

while the C code can be found in

`$SGE_ROOT/util/resource/loadsensors/phi/`.

The load sensor can be tested from the command line. In order to indicate to the load sensor that it has to report new values you must press enter. If this works (either on your self-compiled version or on the pre-compiled you can proceed with the installation of the resource surrogates in Univa Grid Engine's complex configuration.

Resource Configuration for the Intel Phi Load Sensor

Depending on the maximum amount of Xeon Phi cards installed in your system you need to make Univa Grid Engine aware of the new resources. A helper script, which installs the new resources in the complex configuration can be found in

`$SGE_ROOT/util/resource/loadsensors/phi/uge_phi_complex_installer.sh`.

The script can be called in different ways: There is a recommended guided installation when called with `--dialog` (*dialog* utility needs to be installed on a `lx-amd64` admin host), `--install-default <amount of mics>` installs the default complexes for cards, `--remove` removes all installed complex resources (all complexes with the prefix `mic.*.`), `--show-default <amount of mics>` shows the complex entries without actually installing it (this can be useful when the complexes are going to be installed manually or when installation hints should be displayed again). The most flexible option of the script is `--install <complex list> <amount of mic>`, where the complex list is a list of load values which should be reported by the load sensor.

Example:

```
./uge_phi_complex_installer.sh --dialog
```

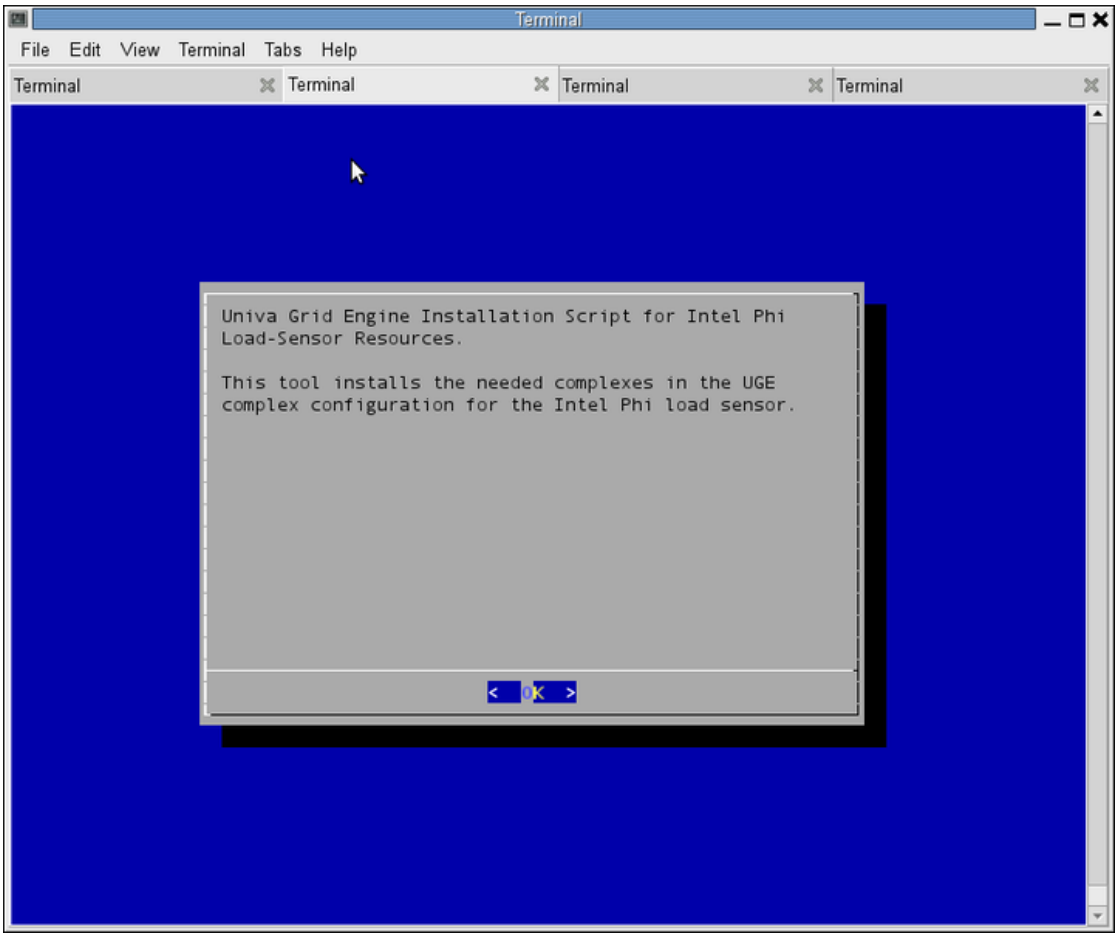


FIGURE: The Intel Xeon Phi Resource Installation Tool

Enter now the maximum amount of Intel Xeon Phi cards installed on a single host (not the cluster total values).

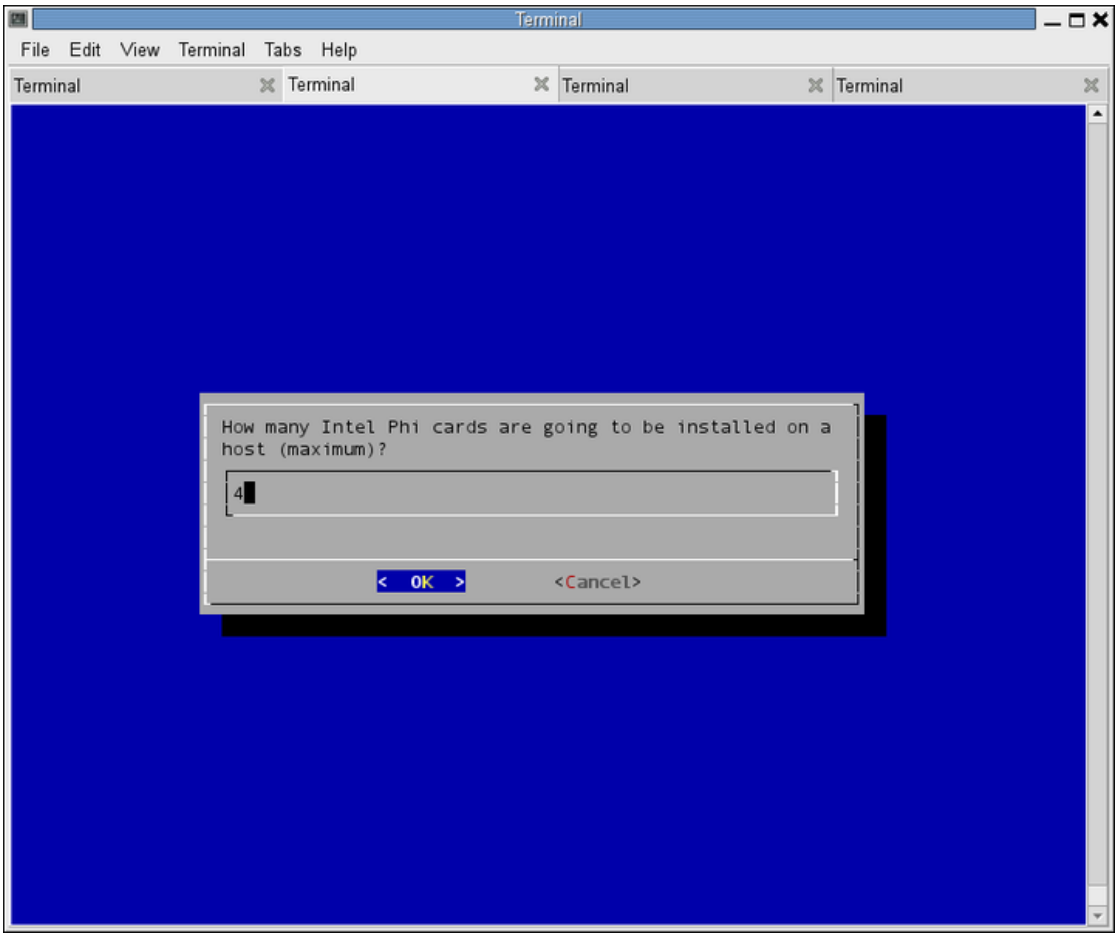


FIGURE: The Intel Xeon Phi Resource Installation Tool

Select now the load values, which should be processed by Univa Grid Engine. Hint: When later more load values should be added to the system, this can be done by removing (`--remove` argument) the installed complexes and re-run the script again. But all existing references, like to the **phi** complex must be deleted first (with `qconf -me <hostname>` or `qconf -dattr ...`).

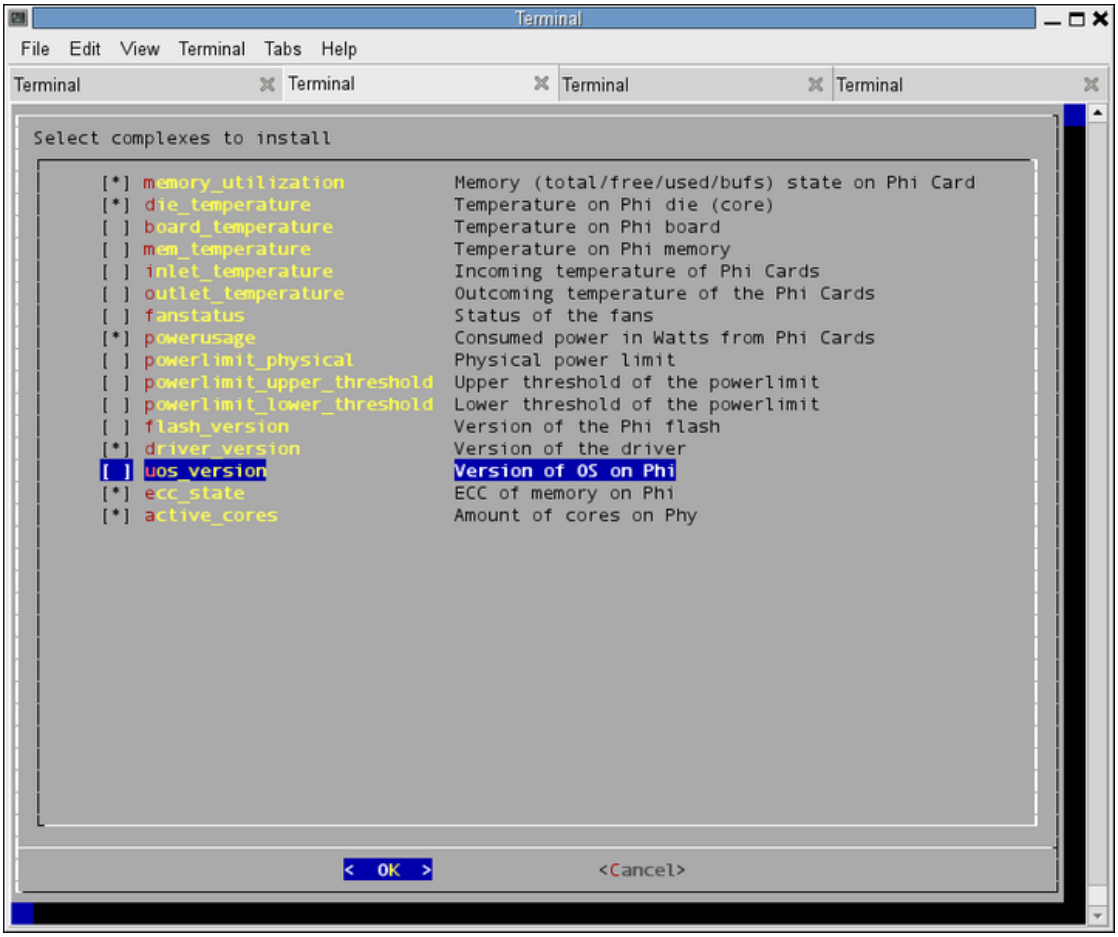


FIGURE: The Intel Xeon Phi Resource Installation Tool

After pressing **ok** the complexes are going to be installed in Univa Grid Engine's complex configuration.

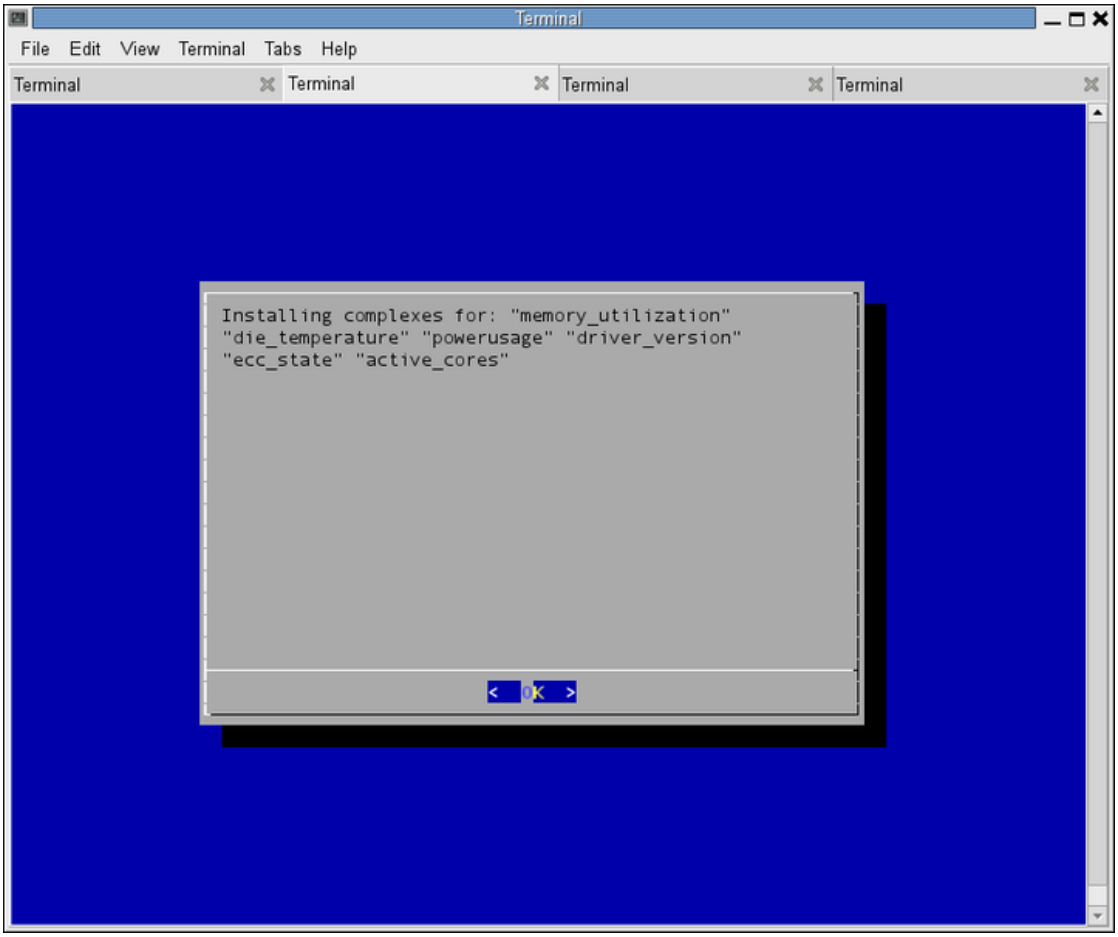


FIGURE: The Intel Xeon Phi Resource Installation Tool

The final output on the terminal shows additional hints for installing the load sensor on the execution host. Important: One section shows the configuration, which is needed for the load sensor. In the example it is following output:

```
active_cores ecc_state driver_version powerusage die_temperature memory_utilization
```

This output is also copied in a file in the current directory with the name `phi_loadsensor_config.txt`. This file can be used as config file for the load sensor, which is explained in the next sub-section.

The installation of the resource complexes can be verified by performing `qconf -sc`. The configuration now must contain several entries with the prefix `mic.<no>..."`.

Installing the Intel Phi Load Sensor

After installing the complexes for the resources, the Intel Xeon Phi load sensor must be installed on the corresponding execution hosts. This is done by simply adding the load sensor in the host configuration with the `qconf -mconf <hostname>`. If there is no load_sensor entry yet in the configuration, following line has to be inserted:

```
load_sensor $SGE_ROOT/util/resources/loadsensors/phi/phi_sensor.sh
```

Where `phi_sensor.sh` script must be must be adapted either to be called with the right parameter or to point to the self-compiled Intel Xeon Phi load sensor. The parameter must be replaced by `-c /path/to/configfile`, i.e. with the path of the configuration file generated in the last sub-section. This file regulates which load values should be reported. When you have installed all complexes, or the accepted the default configuration no config file is needed. Then the load sensor can be called with the -a parameter (reports all load values) or without any parameter (reports default load values).

The installation can be verified with calling the `qhost -F -h <hostname>` command. The new load should be displayed after a while.

In order to configure the amount of Intel Xeon Phi devices per host, the number must be inserted in the host configuration. The installer dialog already added a complex named **phi** for this purpose. The next section shows an example of an advanced configuration, so that each job is automatically bound to the processors near the selected Phi device.

3.14.2 Exploiting the RSMAP topology mask with Intel® Xeon Phi™ Co-Processors

One common issue when having multi-socket hosts with a NUMA architecture is that some PCIe devices are (like memory) closely connected to specific sockets. When an application wants to communicate with the PCIe device, but the application runs on a different socket, additional traffic is sent on the bus between the sockets and additionally the latency for accessing the device is higher.

In order to address this issue, Univa Grid Engine supports since 8.1.3 RSMAP topology masks. The RSMAP complex type is a special consumable, which maps jobs to specific resource IDs (like Intel Phi Co-processor numbers or IP addresses of Intel Phi Cards) to jobs.

The following example demonstrates the usage of RSMAP topology masks in order to bind jobs near a specific Phi device. In the example the host offers 2 Intel Phi devices, known as *MIC0* and *MIC1*, both connected to different sockets. When an application uses one of such a *MIC* device, it should automatically bound to the corresponding socket.

In order to figure out where which device is bound, the `numa_node` or `local_cpulist` entries of the particular device can be used. The /sys/class/mic/mic/ directory contains a link to the corresponding PCI directory. The links has the prefix `pci` followed by the PCI number. Which cores are near to a specific MIC device can be seen by printing out the content of the `local_cpulist` and/or using the `numa_node` file.

```
[daniel@maui mic0]$ cat /sys/class/mic/mic0/pci_*/local_cpulist
0-7,16-23
[daniel@maui mic0]$ cat /sys/class/mic/mic1/pci_*/local_cpulist
8-15,24-31
[daniel@maui mic0]$ cat /sys/class/mic/mic0/pci_*/numa_node
0
[daniel@maui mic0]$ cat /sys/class/mic/mic1/pci_*/numa_node
1
```

The corresponding Univa Grid Engine topology can be seen by calling `$SGE_ROOT/utilbin/loadcheck -cb`.

```
[daniel@maui mic0]$ $SGE_ROOT/utilbin/lx-amd64/loadcheck -cb
Topology:          SCTTCTTCTTCTTCTTCTTCTTSCTTCTTCTTCTTCTTCTTCTT
...
NUMA node of core    0 on socket    0               : 0
NUMA node of core    1 on socket    0               : 0
NUMA node of core    2 on socket    0               : 0
NUMA node of core    3 on socket    0               : 0
NUMA node of core    4 on socket    0               : 0
NUMA node of core    5 on socket    0               : 0
NUMA node of core    6 on socket    0               : 0
NUMA node of core    7 on socket    0               : 0
NUMA node of core    0 on socket    1               : 1
NUMA node of core    1 on socket    1               : 1
NUMA node of core    2 on socket    1               : 1
NUMA node of core    3 on socket    1               : 1
NUMA node of core    4 on socket    1               : 1
NUMA node of core    5 on socket    1               : 1
NUMA node of core    6 on socket    1               : 1
NUMA node of core    7 on socket    1               : 1
```

Here we can see that the all cores on the first socket are on NUMA a node 0, which is the local NUMA node of `mic0`. That means that jobs using `mic0` should run only on cores of socket 0 in order to get the best performance. A topology mask connected to the `mic0` prevents that jobs using `mic0` are running on others cores. A topology mask is a string (like the topology string), which forbids cores by marking them with a "c", while cores which are allowed have a "C". The mask is appended to the RSMAP id by a colon. Topology masks are only allowed for RSMAPs, which are defined as HOST consumables in the complex configuration. Example: The topology mask "ScCScC" allows a job only to run on the second core of the first socket and on the second core of the second socket.

Now the resource complex for the Intel Phi devices have to be generated. This is done by opening the complex configuration, and adding then a new entry. **If the complex installer from the last section was called, the** phi** complex was already added!**

```
qconf -mc
 #name shortcut  type      relop requestable consumable default  urgency
 #-------------------------------------------------------------------------
phi      phi     RSMAP      <=      YES         HOST       0        0
```

The name and shortcut can be arbitrary. Important is that the consumable is type "HOST". A host consumable (new in 8.1.3) decrements a certain job request (-l) just one time per host (no multiplication with slots per host), for all hosts used by the job. It does not depend on the amount of slots, the job get granted on a particular host.

In order to configure the amount of available Intel Phi devices on a host, as well provide the topology mapping, the execution host configuration must be changed. This can be done either

by opening the execution host configuration with `qconf -me maui` or by inserting the new configuration in a single line:

```
qconf -mattr exechost complex_values "phi=2(mic0:SCCCCCCCCCScccccccc
                                      mic1:ScccccccccSCCCCCCCCC)" maui
```

What this does is to make 2 phi resources available on host `maui`, where the first resource is denoted by *mic0* and the second resource by mic1. If the scheduler picks resource mic0, then the job is only allowed to be bound on the first socket, while when it chooses *mic1*, the job is only allowed to run on the second socket.

Submission of jobs requesting a Intel Phi device

Jobs can now be submitted by requesting the amount of Intel Phi devices they need. Since the complex was named `phi`, the job needs to request `-l phi=<amount>`. Due to the configured topology mask, jobs without core binding request are automatically bound to all cores specified by the mask as usable (upper case letters). Jobs requesting a specific amount of cores with a core binding request are only bound to that amount of cores.

```
qsub -b y -l phi=1 <yourjob>

qstat -j <id>
...
binding               1:    maui=0,0:0,1:0,2:0,3:0,4:0,5:0,6:0,7
resource map          1:    phi=maui=(mic0)
```

The job itself can now discover the selected Intel Phi device by reading out the `$SGE_HGR_phi` environment variable, which is set in this case to `mic0`.

When a second job with a core binding request (one core) is submitted the scheduler selects `mic1` and one of the allowed and currently unbound cores.

```
qsub -b y -l phi=1 -binding linear:1 <yourjob>

qstat -j <id>
...
binding               1:    maui=1,0
resource map          1:    phi=maui=(mic1)
```

The jobs `$SGE_HGR_phi` environment variable contains `mic1` and the job is bound to the first core of the first socket.

When checking the host state with the `command`, then all bound/unbound cores can be seen.

```
qhost -F -h <exechost>
  hl:m_topology_inuse=ScttcttcttcttcttcttcttcttScttCTTCTTCTTCTTCTTCTTCTT
```

Info: Since the RSMAP id's can be (almost) arbitrary strings, the corresponding IP address can be configured as well, which makes it easy for the job to login to the right device (like phi=2(172.31.1.1:SCCCCCCCCCScccccccc 172.31.2.1:ScccccccSCCCCCCCC)).

3.14.3 Submission and Directly Starting of an Intel Xeon Phi (MIC) Native Application

The latest version of the Intel Xeon Phi driver comes with an application named **micnativeloadex**, which usually can be found in the */opt/intel/mic/bin* directory. This application can be used in order to startup a native application on a Intel Xeon Phi card selected by Univa Grid Engine. This section shows an example how Univa Grid Engine can be used in order to directly start native applications on appropriate Xeon Phi devices.

Before starting setting up an example job submission and job script, the Intel Xeon Phi devices must be setup correctly. For this the Univa Grid Engine resource configuration (complex configuration) must be opened with

```
qconf -mc
```

Please ensure that the *phi* resource is configured as a **RSMAP** and a per **HOST** consumable.

```
phi             phi           RSMAP       <=    YES         HOST       0        0
```

Now you need to initialize the amount of Xeon Phi cards you have on your host and assign them the CPUs near to that PCIe device (please read the sections above). Because the **micnativeloadex** tool selects the cards based on logical numbers, the **RSMAP** configuration needs to have a number for each card. Following example shows a configuration for 4 Xeon Phi cards on a host.

```
qconf -se maui
hostname              maui
load_scaling          NONE
complex_values        m_mem_free=65494.000000M,m_mem_free_n0=32726.765625M, \
                      m_mem_free_n1=32768.000000M,phi=4(0:SCCCCccccScccccccc \
                      1:SccccCCCCScccccccc 2:SccccccccSCCCCcccc \
                      3:SccccccccScccCCCC)
```

When a *phi* resource is selected it gets a number from 0 to 3 in the `$SGE_HGR_phi` environment variable which is used to direct the access to the appropriate card.

The following job script starts the Xeon Phi native application directly on the card. The job itself has set an `$SINK_LD_LIBRARY_PATH` which is evaluated by the **micnativeloadex** tool and forwarded as `$LD_LIBRARY_PATH` to the application on the Xeon Phi card. The `$MIC_APP`, `$MIC_TIMEOUT`, `$MIC_ARGS`, and `$MIC_ENV environment` variables are also set by the job submission script during job submission time. The *-t* parameter sets a timeout. When the application is not finished in `$MIC_TIMEOUT` seconds, the it is aborted and gets deleted from the device by the Intel tool. `$MIC_APP` is the path to the native application, which must exist on the execution host (usually placed on a shared filesystem).

```
cat example_job.sh

#!/bin/sh
/opt/intel/mic/bin/micnativeloadex $MIC_APP -t $MIC_TIMEOUT -v -d $SGE_HGR_phi
-a "$MIC_ARGS" -e "$MIC_ENV"
```

The job submission template is the script the user must adapt in order to let his own native application run.

```
cat example_submission.sh

#!/bin/sh
# The maximum runtime of the application (enforced limit).
# Note: The MIC native application is terminated 30 seconds before,
# if it not finished at that point in time.
JOB_RUNTIME_SEC=120
# Put here the path to the MIC native binary.
MIC_APP=/opt/intel/mic/perf/bin/mic/stream_mic
# Put here the command line arguments for the MIC native binary.
MIC_ARGS=""
# The environment variables which should be set for the MIC
# native binary. Put here one or more space separated environment
# variables which should be set in the MIC application.
MIC_ENV="e1=test1 e2=test2"
# This path is the LD_LIBRARY_PATH on the MIC. Put in here
# all your libraries the MIC application needs.
SINK_LD_LIBRARY_PATH="/opt/intel/lib/mic"
# Request a runtime for the job, an Intel Xeon Phi card
# and forward the job etc. as environment variables to
# the job.
qsub -l h_rt=$JOB_RUNTIME_SEC,phi=1 -v MIC_TIMEOUT=`expr $JOB_RUNTIME_SEC - 30`,
MIC_APP=$MIC_APP,MIC_ARGS="$MIC_ARGS",MIC_ENV="$MIC_ENV",SINK_LD_LIBRARY_PATH=
"$SINK_LD_LIBRARY_PATH" example_job.sh
```

The **h_rt** parameter will let Univa Grid Engine kill the job script on the host after *$JOB_RUNTIME_SEC* seconds. From this runtime value 30 seconds are decremented, which is the maximum runtime for the Xeon Phi native application on the cards. If the Xeon Phi native jobs does not end before that time, it is cleaned up by the Intel tool.

Note
It might be possible that the micnativeloadex application changes over time and the scripts must be adapted.

3.14.4 The mic_load_check tool

If an application wants to access load metrics of the Intel Xeon Phi cards directly the **mic_load_check** utility can be used. Univa Grid Engine ships this example tool which prints out metrics like amount of installed cards, or load status information of specific cards on command line. Like for the Intel Xeon Phi load sensor the C code is also available so that it can be easily adapted for specific purposes or re-compiled for other architectures than RHEL 6.3. The tool is located in *$SGE*ROOT/util/resources/loadsensors/phi/lx-amd64_. It needs a running MPSS service (shipped with the Intel Xeon Phi cards) as well as the MIC Intel SDK library (libMicAccessSDK.so) in the library path. More information is available in the help output of *mic*load_check_, when called with the **-h** parameter.

3.14.5 Configure and Install Intel Xeon Phi x200 (Knights Landing) Processors support

The Intel Xeon Phi x200 (Knights Landing) is the second generation MIC architecture from Intel. The x200 is a bootable and standalone host processor with up to 72 cores and 16 of HBM. The cores are structured in a mesh of rings where every row and column builds a ring.

The x200 can run in 3 different so called Cluster Modes and 3 different Memory Modes.

Cluster Mode	Description
All-to-All	Address uniformly hashed across all distributed directories
Quadrant	Chip divided into four virtual Quadrants
Sub-NUMA-Clustering	(SNC) Each Quadrant (cluster) exposed as a separate NUMA domain to OS

Memory Mode	Description
Flat MCDRAM	MCDRAM exposed as a separate NUMA Node and is explicitly allocatable
MCDRAM as Cache	MCDRAM covers all DDR
Hybrid Model	Mixture of Flat and Cache. Parts will be exposed as NUMA Node and parts as DDR e.g. Hybrid50 means 50 % of the MCDRAM are used as separate NUMA Node and 50 % covers all DDR

3.14.6 Installing Intel Xeon Phi x200 (Knights Landing) Processors support

The installer `uge_knl_complex_installer.sh` is located in the 'util/resources/loadsensors/' directory of `$SGE_ROOT`. The installer will add 4 new complexes to Univa Grid Engine:

Complexes	Description
knl_cluster_mode	The cluster mode of the x200 machine
knl_memory_mode	The memory mode of the x200 machine
knl_MCDRAM_total	The MCDRAM which is exposed as a separate NUMA Node
knl_MCDRAM_cache	The MCDRAM which is used as DDR

By default it will also add the pre-compiled 'load_sensor' to the configuration of the given host.

Usage of the installer:

Parameter	Description
–install hostname	Add all complexes to Univa Grid Engine and add the 'load_sensor' to the given host.
–install_only_loadsensor hostname	Add only the 'load_sensor' to the given host.
–remove_all hostname	Remove the complexes from Univa Grid Engine as also the 'load_sensor' from the given host.
–remove_complexes	Remove the complexes from Univa Grid Engine.
–remove_loadsensor hostname	Remove the 'load_sensor' from the given host.

3.15 Integration with Docker® Engine

Univa Grid Engine provides an integration with Docker Engine that allows to run jobs in Docker containers. This integration is implemented only on Linux execution hosts and needs to have the Docker Engine properly installed on that execution host.

Univa Grid Engine automatically detects if Docker Engine is installed on an execution host. If Docker Engine of a supported version is installed and running on an execution host, Univa Grid Engine reports the value `1` for the host load value `docker`.

The Docker images that are available on that execution host are reported using the host load value `docker_images`. This load value is one string of type `RESTRING` that contains all available Docker images, separated by commas. This type was selected because there is no better one available in Univa Grid Engine currently, a kind of string list would be more adequate, see the example below why. After the execution daemon start, this load value takes at least one `load_report_time` longer than all other load values to be reported, depending on the Docker daemon it can be even longer.

There are corresponding complex variables defined which are automatically set by the load values, so a job can to be started in a Docker container by requesting both the "docker" resource and a specific Docker image to use. Because the `docker_images` variable is of type `RESTRING`, this request must be a regular expression, e.g.:

```
-l docker_images="*ubuntu:14.04*"
```

Without the asterisks, the request would match only if there is just the `ubuntu:14.04` image available, but no others. A full job submit command line of a Docker job looks like this:

```
> qsub -l docker,docker_images="*ubuntu:14.04*" -S /bin/sh -b y hostname
```

If Docker is installed on an execution host, but the "docker" host load value is reported as `0`, there are several reasons for this.

One reason could be the Docker Engine version is not supported. Supported are Docker Engine versions from 1.8.3 to 1.13.0 and 17.03 to 17.09, on OpenSUSE and SLES up to 1.12.6.

Another reason can be the Docker daemon did not start properly. It has been observed that after installation of the Docker Engine or after a reboot of the execution host the Docker daemon does

not start properly. The service stays in `starting` state and never becomes `running`. If run as root, the command

```
> systemctl status docker
```

shows the status of the service. To fix this, the service has to be restarted manually as root:

```
> systemctl restart docker
```

If Docker is not detected automatically, it doesn't help to overwrite the load value with a configured complex value. If this done, the job will be dispatched to the execution host, but cannot be started and will fail.

On the other hand, an execution host with running Docker Engine can be disabled for Docker jobs by configuring `docker=0` in the `complex_values` list of the execution host. This overwrites the load value, so no Docker jobs get scheduled to this execution host anymore.

Univa Grid Engine allows to submit jobs with the soft request for a Docker image. In this case, Univa Grid Engine tries to find a matching execution host where this image is already available. If it is not available, Univa Grid Engine selects and matching execution host with Docker Engine installed and running and tells the Docker daemon to download the image before job start.

It has been observed that this downloading is broken with some Linux distributions and some Docker versions. In this case, the Docker images have to be downloaded manually to that execution host using the `docker run` command, e.g.:

```
> docker run notavailableimage:latest hostname
```

will load the latest version of the `notavailableimage`.

3.15.1 Docker images suitable for autostart Docker jobs with arguments

The so called autostart Docker jobs do not specify a job script or binary to start, instead they use the keyword `NONE` to indicate the Docker container shall be started by running the script or binary that is defined in the `ENTRYPOINT` of the Docker image.

The `ENTRYPOINT` can be viewed by running

```
$ docker inspect image:tag
```

relevant is the data in the section `Config`. When the Docker image is built it is possible to specify the `ENTRYPOINT` in the `Dockerfile` in an informal way, e.g.:

```
ENTRYPOINT /path/to/script
```

The `docker build` command will accept this, but to such an image, no arguments can be specified (while both this might change with the exact Docker version). If the `ENTRYPOINT` is specified properly, the arguments on the commandline are forwarded to the script or binary defined in the `ENTRYPOINT`, e.g.:

```
ENTRYPOINT ["/path/to/script"]
```

It is also possible to define arguments that always are provided to the script or binary, e.g.:

```
ENTRYPOINT ["/path/to/script", "fixedarg1", "fixedarg2"]
```

The script or binary in the Docker container created from this image will always get `fixedarg1` as first argument and `fixedarg2` as second argument. If there are further arguments specified on the command line, they are appended to the "fixed" ones. E.g. if the script just prints the arguments it gets, this command line would produce the output:

```
$ docker run -it myimage:latest arg1 arg2
fixedarg1 fixedarg2 arg1 arg2
```

or with a Univa Grid Engine job:

```
$ qsub -l docker,docker_images="*myimage:latest*" -b y NONE arg1 arg2
$ cat ~/NONE.o1
fixedarg1 fixedarg2 arg1 arg2
```

Additionally, the `CMD` and `RUN` entries can affect the behaviour of the Docker container and could prevent the provided arguments from being forwarded properly.

3.15.2 Run the container as root, allow to run prolog etc. as a different user

With the `execd_params START_CONTAINER_AS_ROOT` it is now possible to let all Docker containers be started as root and allow the prolog, pe_start, per_pe_task_prolog, per_pe_task_epilog, pe_stop and epilog scripts to be started as a different user than the job owner. This change applies not to "autostart Docker jobs", i.e. jobs that specify `-b y NONE` as job script in order to use the entrypoint that is defined in the Docker image instead of using the `sge_container_shepherd` as entrypoint.

3.15.3 Automatically map user ID and group ID of a user into the container

If the `START_CONTAINER_AS_ROOT` parameter is set to `true`, it is now necessary that the Univa Grid Engine admin user, the job user and all pre and post script users are defined inside the container. Because this is usally not the case, by setting the `AUTOMAP_CONTAINER_USERS` parameter to `TEMPORARY`, Univa Grid Engine transfers the user ID and group ID of any of these users from the host to the container. But only Univa Grid Engine itself can use this information there, it is not available for the job or any of the scripts started by Univa Grid Engine!

If `AUTOMAP_CONTAINER_USERS` is set to `PERSISTENT`, Univa Grid Engine writes an entry to the `/etc/passwd` file inside the Docker container for all these users. This allows to lookup the user information in a script, but it does not allow to switch to this user!

Caution! If `AUTOMAP_CONTAINER_USERS=PERSISTENT` is specfied, if an user maps the `/etc/passwd` and `/etc/group` file into the container, the host files are modified!

3.15.4 Create a container_pe_hostfile with all container hostnames

If a parallel Docker job is started where the container hostnames are selected from RSMAPs, the execution daemon of the master task writes a `container_pe_hostfile` with all the container hostnames in the `pe_hostfile` format if the `execd_params CONTAINER_PE_HOSTFILE_COMPLEX` is set to the name of the RSMAP complex that defines the hostnames.

E.g.: If there is a RSMAP "cont_hosts" declared and on each execution host it defines values like:

```
cont_host=4(host1_cont1 host1_cont2 host1_cont3 host1_cont4)
```

and a job is submitted using

```
# qsub -pe mype 4 -l docker,docker_images="*image:latest*",cont_host=1 job_script.sh
```

and the scheduler decides to schedule the master task to host1, two slave tasks to host2 and one slave task to host3, the `container_pe_hostfile` might contain:

```
host1_cont3 1 <NULL> <NULL>
host2_cont1 1 <NULL> <NULL>
host2_cont4 1 <NULL> <NULL>
host3_cont2 1 <NULL> <NULL>
```

This allows to read this information in a per_pe_task_prolog and set the hostnames of the containers inside of the containers accordingly.

3.15.5 Run tightly integrated parallel jobs in Docker containers

What makes tightly integrated parallel jobs different from sequential ones is the fact that usually the master task of the job interacts with Univa Grid Engine in order to start the slave tasks. While sequential jobs also might have certain requirements to their environment and thus cannot run in a not prepared Docker container, for parallel jobs even the start can fail if the Docker container is not setup properly.

A tightly integrated parallel job consists of a master task and several slave tasks. While Univa Grid Engine reserves the slots and other resources for the slave tasks, usually the master task starts the slave tasks using `qrsh -inherit <hostname>`.

Because a parallel Docker jobs runs in a container that - by default - has its own separate Docker network which is not part of the cluster network and that does not know the hostnames and IP addresses of the other cluster hosts, the master task running in this container cannot submit slave tasks to the "physical" slave execution hosts.

In order to be able to do this, inside the container this must be set properly:

- cluster network access
- hostname resolving
- Univa Grid Engine admin user, job user, prolog user, etc. and their respective groups

- Univa Grid Engine CSP certificates, if CSP mode is selected

And, in order to allow the containers to communicate to each other, e.g. for the MPI integration, the container must have a hostname and IP address that is known to other containers.

1) Allow the container to participate in the cluster network:
   The easiest way to achieve this should be using the submit option `-xd "--net=host"` or `-xd "--network host"` (`-xd` means "external docker option" and allows to forward several `docker run` options to the execution host, see `qsub -xd "--help"` for a list of supported options). This makes the container inherit both the hostname and the IP address of the "physical" host.

2) Allow the container to resolve the names of physical hosts or other containers:
   This can be achieved by mapping the `/etc/hosts` file of the physical host in the container by specifying `-xd "-v /etc/hosts:/etc/hosts"` or by creating a temporary hosts file from LDAP/NIS/etc. and mapping this into the container.

3) Define the various users inside the container:
   If the container is started as root by configuring the execd_params `START_CONTAINER_AS_ROOT=TRUE`, the `sge_container_shepherd` will try to start start and stop scripts like prolog as the configured user and will do all file operations as the Univa Grid Engine admin user. Because Univa Grid Engine transfers the user name, not the user ID, inside the container the user name must be known and the user ID must be defined in order to work properly. The user and group names and IDs can be defined in the container by specifying `-xd "-v /etc/passwd:/etc/passwd"` and `-xd "-v /etc/group:/etc/group"` on the qsub command line. Again, instead of using these files from the physical host, temporary files can be created from LDAP/NIS/etc. to be mapped into the container.
   Of course, it is also possible to configure the Docker images to use LDP/NIS/etc. directly. See the Docker documentation for details about this.

4) Univa Grid Engine CSP certificates for CSP mode:
   If the Univa Grid Engine CSP mode is used, the certificates must be mapped into the container in order to allow the master task use `qrsh -inherit`. This is done by specifying `-xd "-v $PATH_TO_CERTS:$PATH_TO_CERTS"` on the qsub or qrsh command line.

5) Allow containers of a MPI Docker job to communicate to each other:
   In order to allow a slave task running in one container to communicate with the slave task in another container using MPI, the container hostnames must be set to known values.
   To achieve this, declare a RSMAP complex and define hostname like described in "Create a container_pe_hostfile with all container hostnames". The container hostnames and IPs must be defined in DNS or `/etc/hosts` and be mounted to all containers as described in 2). Then, the prolog script must move away the `pe_hostsfile` and rename the `container_pe_hostfile` to `pe_hostfile`. Thus, MPI knows the names of the containers the slave tasks run in and can communicate with them.

Examples:
All examples use the job script `job.sh` which uses `qrsh -inherit $HOSTNAME ...` to start the slave tasks which run the task script `task.sh`.

```
$ cat job.sh
   #!/bin/sh

   # set PATH to "qrsh" binary
   export PATH=$SGE_O_PATH:$PATH

   qrsh -inherit -noshell -nostdin $HOSTNAME /home/user/task.sh $1 &
   qrsh -inherit -noshell -nostdin $HOSTNAME /home/user/task.sh $1 &

   exit 0
```

$HOSTNAME is set by UGE to the name of the physical host, not the container hostname, also inside of the container.

```
$ cat task.sh
   #!/bin/sh

   echo "I am a task, it is now `date`! Now sleeping for $1 s."
   sleep $1
   echo "I am a task, it is now `date`! I'm done now."

   exit 0
```

The parallel environment `docker.pe` uses the allocation_rule `$pe_slots`.

Example A)
A parallel Docker job with the container running as the job user (`START_CONTAINER_AS_ROOT=false`)

One could think this is sufficient:

```
$ qsub -pe docker.pe 3 -l docker,docker_images="*ubuntu:14.04*" job.sh 10
```

but then the `qrsh` command inside the job script cannot get information about the job user it is started as and fails. To fix this, add the option `-xd "-v /tmp/my_passwd:/etc/passwd"` which maps the `/tmp/my_passwd` file that must contain information about the job user. This file can either be the `/etc/passwd` file on the host, or a file being generated from LDAP/NIS etc. before job start.

As the next step, the `qrsh` command inside the job script tries to resolve the host it is started on. By default, Docker set the hostname of the container to the first twelve characters of the container ID, which isn't a known hostname in the cluster. It is possible to tell Docker to let the container inherit the hostname and IP address of the host by specifying the `-xd "--net=host"` option.

Furthermore it is necessary to mount the directory the `task.sh` script is located on to the container. In our example, it is located on `/home/user`, so the `/home` directory must be mounted to the container, too.

These two enhancements lead to the submit command line:

```
$ qsub -pe docker.pe 3 -l docker,docker_images="*ubuntu:14.04*" \
       -xd "-v /tmp/my_passwd:/etc/passwd","-v /home:/home","--net=host" \
       ~/job.sh 10
```

With this, our parallel test job now runs properly. This works only if all tasks of a parallel job run on the same host, i.e. if the allocation_rule is `$pe_slots` or `<int>`.

If the slave tasks are distributed over different hosts, the job.sh script has to parse the `pe_hostfile` and submit the `qrsh -inherit $host ...` to respective hosts.

Example B)
A parallel Docker job with the container running as root (`START_CONTAINER_AS_ROOT=true`) that is distributed over several execution hosts

Sometimes it is necessary to run start and stop scripts like the prolog as root in the container in order to setup certain things. With the execd_params config option `START_CONTAINER_AS_ROOT=true`, all containers are started as root. The start and stop scripts can be run as the configured users then, the job itself still is started as the job user.

In order to be able to run the prolog etc. as the configured user, it is necessary to make these users known inside the container. There is the `AUTOMAP_CONTAINER_USERS=PERSISTENT` execd_param that maps the necessary users into the container.

If these two configurations are made, this command line is sufficient to run a parallel Docker job on one host (i.e. allocation_rule is `$pe_slots` or `<int>`):

```
$ qsub -pe docker.pe 3 -l docker,docker_images="*ubuntu:14.04*" \
       -xd "-v /home:/home","--net=host" job.sh 10
```

Again, to start slave tasks on different hosts, the job.sh script has to parse the `pe_hostfile` and submit the `qrsh -inherit $host ...` to these hosts.

3.15.6 Configuring the Docker daemon response timeout

The `execd_params DOCKER_RESPONSE_TIMEOUT` allows to define the time Univa Grid Engine waits for a response from the Docker daemon to a request Univa Grid Engine sent to the Docker daemon before. This does not mean the full resonse must be received within the timeout; the timeout counter is reset after each character Univa Grid Engine receives from the Docker daemon in response to a specific request.

If this parameter is not specified, the default value of 60 s is used. The minimum timeout is 10 s, the maximum timeout is 86400 s. If `DOCKER_RESPONSE_TIMEOUT` is not within this range, the default value is used instead.

3.16 Special Tools

3.16.1 The Loadcheck Utility

The `loadcheck` utility is located in the 'utilbin' directory of `$SGE_ROOT`. It retrieves and shows load values of the host, where it is started. It shows the number of detected processors, the execution host topology (if it can be retrieved), and CPU/memory load values.

```
../utilbin/lx-amd64> ./loadcheck
arch            lx-amd64
num_proc        1
m_socket        1
m_core          1
m_thread        1
m_topology      SC
load_short      0.07
load_medium     0.14
load_long       0.07
mem_free        1532.828125M
swap_free       2053.996094M
virtual_free    3586.824219M
mem_total       1960.281250M
swap_total      2053.996094M
virtual_total   4014.277344M
mem_used        427.453125M
swap_used       0.000000M
virtual_used    427.453125M
cpu             0.0%
```

The default format for the memory values can be turned into an integer format with the parameter `-int`. Additionally it has a build-in debugging facility for obtaining more details about the execution host topology and the core binding feature. When the application is called with the `-cb` switch, it prints out internal kernel statistics (on Solaris) and on Linux the mapping of socket/core numbers to the internal processor ID.

```
> ./loadcheck -cb
Your UGE Linux version has built-in core binding functionality!
Your Linux kernel version is: 2.6.34.7-0.7-desktop
Amount of sockets:     1
Amount of cores:       1
Amount of threads:     1
Topology:          SC
Mapping of logical socket and core numbers to internal
Internal processor ids for socket     0 core     0:      0
```

3.16.2 Utilities for BDB spooling

Univa Grid Engine can be configured to use a Berkeley DB for spooling in sge_qmaster.

Berkeley DB comes with a number of commandline tools, some of which can be useful for operating and debugging Univa Grid Engine spooling.

Warning

Do not use these tools on a database which is in use by an active sge_qmaster. Only use the tools when advised to do so by a support engineer.

The following tools are delivered with Univa Grid Engine:

- db_deadlock: Deadlock detection utility.
- db_dump: Database dump utility - used for doing backup by `inst_sge -bup`
- db_load: Database load utility - used for restoring a database dump, called by `inst_sge -rst`
- db_printlog: Transaction log display utility.
- db_recover: Recovery utility.
- db_stat: Statistics utility.
- db_upgrade: Database upgrade utility.
- db_verify: Verification utility.

The full Berkeley DB documentation for these tools is part of the Univa Grid Engine distribution in the common package. To access it, view `$SGE\_ROOT/doc/berkeleydb/utility/index.html` with a web browser.