

Table of 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.....	6
1.1.2.1 Displaying Job Status Information.....	6
1.1.2.2 Understanding the Various Job States.....	6
1.1.2.3 Displaying Host Status Information.....	8
1.1.2.4 Understanding the Various Host States.....	9
1.1.2.5 Displaying Queue Status Information.....	11
1.1.2.6 Understanding the Various Queue States.....	11
1.2 Understanding a Default Installation.....	12
1.2.1 Default Queue.....	12
1.2.2 Default PE.....	13
1.2.3 Default User Set Lists.....	13
1.2.4 Default Host Group List.....	14
1.2.5 Default Complex Attributes.....	14
1.3 Understanding Key Univa Grid Engine Configuration Objects.....	15
1.3.1 The Cluster Configuration.....	15
1.3.2 The Scheduler Configuration.....	15
1.3.3 Host and Queue Configurations.....	16
1.4 Navigating the ARCo Database.....	16
1.4.1 Accessing the ARCo Database.....	16
1.4.2 Views to the Database.....	16
1.4.2.1 Accounting.....	17
1.4.2.1.1 view accounting.....	17
1.4.2.2 Job related Information.....	18
1.4.2.2.1 view job log.....	18
1.4.2.2.2 view job times.....	19
1.4.2.2.3 view jobs completed.....	20
1.4.2.3 Advance Reservation Data.....	21
1.4.2.3.1 view ar attribute.....	21
1.4.2.3.2 view ar log.....	22
1.4.2.3.3 view ar resource usage.....	22
1.4.2.3.4 view ar time usage.....	22
1.4.2.3.5 view ar usage.....	23
1.4.2.4 Values Related to Univa Grid Engine Configuration Objects.....	23
1.4.2.4.1 view department values.....	24
1.4.2.4.2 view group values.....	24
1.4.2.4.3 view host values.....	25
1.4.2.4.4 view project values.....	25
1.4.2.4.5 view queue values.....	26
1.4.2.4.6 view user values.....	27

Table of Contents

1 Navigating and Understanding

1.4.2.5 Statistics.....	27
1.4.2.5.1 view_statistic.....	27
1.4.3 Database Tables.....	28
1.4.3.1 Job Data and Accounting.....	29
1.4.3.1.1 sge job.....	29
1.4.3.1.2 sge job log.....	30
1.4.3.1.3 sge job request.....	30
1.4.3.1.4 sge job usage.....	31
1.4.3.2 Advance Reservation Data.....	32
1.4.3.2.1 sge ar.....	32
1.4.3.2.2 sge ar attribute.....	32
1.4.3.2.3 sge ar log.....	33
1.4.3.2.4 sge ar resource usage.....	33
1.4.3.2.5 sge ar usage.....	33
1.4.3.3 Values Related to Univa Grid Engine Configuration Objects.....	34
1.4.3.3.1 sge department.....	34
1.4.3.3.2 sge group.....	34
1.4.3.3.3 sge host.....	34
1.4.3.3.4 sge project.....	35
1.4.3.3.5 sge queue.....	35
1.4.3.3.6 sge user.....	35
1.4.3.3.7 sge department values.....	36
1.4.3.3.8 sge group values.....	36
1.4.3.3.9 sge host values.....	36
1.4.3.3.10 sge project values.....	37
1.4.3.3.11 sge queue values.....	37
1.4.3.3.12 sge user values.....	38
1.4.3.4 Sharetree Usage.....	38
1.4.3.4.1 sge share log.....	38
1.4.3.5 Statistics.....	39
1.4.3.5.1 sge statistic.....	39
1.4.3.5.2 sge statistic values.....	39
1.4.3.6 dbwriter Internal Data.....	40
1.4.3.6.1 sge checkpoint.....	40
1.4.3.6.2 sge version.....	40

2 Common Tasks.....41

2.1 Common Administrative Tasks in a Univa Grid Engine System.....	41
2.1.1 Draining Then Stopping the Cluster.....	41
2.1.2 Starting Up and Activating Nodes Selectively.....	42
2.1.3 Adding New Execution Hosts to an Existing Univa Grid Engine System....	42

Table of Contents

2 Common Tasks

2.1.4	<u>Generate/Renew Certificates and Private Keys for Users</u>	44
2.1.5	<u>Backup and Restore the Configuration</u>	45
2.1.5.1	<u>Creating a Manual Backup</u>	45
2.1.5.2	<u>Automating the Backup Process</u>	46
2.1.5.3	<u>Restoring from a Backup</u>	47
2.2	<u>Managing User Access</u>	48
2.2.1	<u>Setting Up a Univa Grid Engine User</u>	49
2.2.2	<u>Administrators</u>	49
2.2.3	<u>Operators and Owners</u>	50
2.2.4	<u>User Access Lists and Departments</u>	51
2.2.4.1	<u>Commands to Add, Modify Delete Access Lists</u>	51
2.2.4.2	<u>Configuration Parameters of Access Lists</u>	52
2.2.5	<u>Projects</u>	52
2.2.5.1	<u>Commands to Add, Modify Delete Projects</u>	52
2.2.5.2	<u>Configuration Parameters of Projects</u>	53
2.3	<u>Understanding and Modifying the Cluster Configuration</u>	53
2.3.1	<u>Commands to Add, Modify, Delete or List Global and Local Configurations</u>	53
2.3.2	<u>Configuration Parameters of the Global and Local Configurations</u>	54
2.4	<u>Understanding and Modifying the Univa Grid Engine Scheduler Configuration</u> ..	55
2.4.1	<u>The Default Scheduling Scheme</u>	56
2.5	<u>Configuring Properties of Hosts and Queues</u>	58
2.5.1	<u>Configuring Hosts</u>	58
2.5.1.1	<u>Local Cluster Configuration</u>	58
2.5.1.2	<u>Execution Host Configuration</u>	59
2.5.1.3	<u>Administrative and Submit Hosts</u>	61
2.5.1.4	<u>Grouping of Hosts</u>	61
2.5.1.5	<u>Example: Grouping Host-Groups in a Tree Structure</u>	62
2.5.2	<u>Configuring Queues</u>	63
2.5.2.1	<u>Example: Adding a New Queue, Showing the Queue Configuration and Deleting the Queue</u>	64
2.5.2.2	<u>Queue Configuration Attributes</u>	65
2.5.2.2.1	<u>Queue Limits</u>	65
2.5.2.2.2	<u>Queue Sequencing and Thresholds</u>	66
2.5.2.2.3	<u>Queue Checkpoints, Processing and Type</u>	66
2.5.2.2.4	<u>Queue Scripting</u>	67
2.5.2.2.5	<u>Queue Signals and Notifications</u>	68
2.5.2.2.6	<u>Queue Access Controls and Subordination</u>	68
2.5.2.2.7	<u>Queue Complexes</u>	68
2.5.2.2.8	<u>Queue Calendar and State</u>	68
2.5.3	<u>Utilizing Complexes and Load Sensors</u>	69

Table of Contents

2 Common Tasks

2.5.3.1	Configuring Complexes.....	69
2.5.3.1.1	Adding, Modifying and Deleting Complexes.....	69
2.5.3.1.2	Initializing Complexes.....	71
2.5.3.1.3	Using Complexes.....	72
2.5.3.2	Configuring Load Sensors.....	73
2.5.4	Advanced Attribute Configuration.....	73
2.5.4.1	Example: Modification of a Queue Configuration.....	74
2.6	Monitoring and Modifying User Jobs.....	75
2.7	Diagnostics and Debugging.....	76
2.7.1	Diagnosing Scheduling Behavior.....	76
2.7.2	Location of Logfiles and Interpreting Them.....	77
2.7.3	Turning on Debugging Information.....	78
2.7.3.1	Activating Scheduler Profiling.....	78
2.7.3.2	Activating Scheduler Monitoring.....	80
2.7.3.2.1	Find Reasons Why Jobs are Not Started.....	81
2.7.3.2.2	Enable Monitoring to Observe Scheduler Decisions.....	81
2.7.3.3	Activate Debugging Output from the Command-Line and How to Interpret It.....	83
2.7.3.4	Using DTrace for Bottleneck Analysis.....	83

3 Special Activities.....86

3.1	Tuning Univa Grid Engine for High Throughput.....	86
3.1.1	sge qmaster Tuning.....	86
3.1.1.1	Setup Options.....	86
3.1.1.2	Configuration Options.....	86
3.1.2	Tuning Scheduler Performance.....	87
3.1.3	Reducing Overhead on the Execution Side.....	88
3.1.3.1	Local sge_execd Spooling.....	88
3.1.3.2	Switch off PDC.....	88
3.1.4	Choosing Job Submission Options.....	88
3.2	Tuning Univa Grid Engine for Large Parallel Applications.....	89
3.2.1	General Settings.....	89
3.2.1.1	Interactive Job Support.....	89
3.2.1.2	Accounting Summary.....	89
3.2.2	Tuning on the Execution Side.....	90
3.2.2.1	Tuning sge_execd.....	90
3.2.2.2	Reducing Impact of sge_execd on the Execution Host.....	90
3.2.3	Job Related Tuning.....	91
3.3	Optimizing Utilization.....	91
3.3.1	Using Load Reporting to Determine Bottlenecks and Free Capacity.....	91
3.3.2	Scaling the Reported Load.....	94

Table of Contents

3 Special Activities

3.3.2.1	Example: Downscale load short by a Factor of 10.....	94
3.3.3	Alternative Means to Determine the Scheduling Order.....	95
3.3.3.1	Queue Sequence Number.....	95
3.3.3.1.1	Example: Defining the Queue Order.....	95
3.3.3.1.2	Example: Defining the Order on Queue Instance Level.....	96
3.3.3.1.3	Example: Antipodal Sequence Numbering of Queues.....	97
3.4	Managing Capacities.....	97
3.4.1	Using Resource Quota Sets.....	97
3.4.2	Using Consumables.....	100
3.5	Implementing Preemption Logic.....	104
3.5.1	When to Use Preemption.....	104
3.5.2	Utilizing Queue Subordination.....	104
3.5.2.1	Example: Suspend all low priority jobs on a host whenever a job is running in the high priority queue.....	105
3.5.3	Advanced Preemption Scenarios.....	105
3.5.3.1	Example: Mixing exclusive high priority jobs with low priority jobs.....	106
3.6	Integrating Univa Grid Engine With a License Management System.....	107
3.6.1	Integrating and Utilizing QLICSERVER.....	108
3.7	Managing Priorities and Usage Entitlements.....	109
3.7.1	Fair-Share (Share Tree) Ticket Policy.....	109
3.7.1.1	Halftime and Compensation Factor.....	112
3.7.2	Functional Ticket Policy.....	113
3.7.3	Override Ticket Policy.....	114
3.7.4	Urgency Policy.....	115
3.7.4.1	Wait Time Urgency.....	115
3.7.4.2	Deadline Urgency.....	115
3.7.4.2.1	Example.....	116
3.7.4.3	Resource-Dependent Urgencies.....	116
3.7.5	User Policy: POSIX Policy.....	117
3.7.5.1	Example.....	117
3.8	Advanced Management for Different Types of Workloads.....	118
3.8.1	Parallel Environments.....	118
3.8.1.1	Commands to Configure Parallel Environment Object.....	119
3.8.1.2	Configuration Parameters of Parallel Environments.....	119
3.8.1.3	Setup Parallel Environment for PVM Jobs.....	123
3.8.1.4	Submitting Parallel Jobs.....	124
3.8.2	Setting Up Support for Interactive Workloads.....	125
3.8.3	Setting Up Support for Checkpointing Workloads.....	125
3.8.3.1	Commands to Configure Checkpointing Environments.....	125
3.8.3.2	Configuration Parameters for Checkpointing Environments.....	126
3.8.4	Enabling Reservations.....	128

Table of Contents

3 Special Activities

3.8.4.1	Reservation and Backfilling.....	128
3.8.4.2	Advance Reservation.....	130
3.8.5	Simplifying Job Submission Through the Use of Default Requests.....	133
3.8.6	Job Submission Verifiers.....	134
3.8.6.1	Using JSVs for Ensuring Correctness of Job Submissions.....	134
3.8.6.1.1	Locations to Enable JSV.....	135
3.8.6.1.2	JSV Language Support.....	136
3.8.6.1.3	JSV Script Interface Functions.....	137
3.8.6.1.4	Parameter Names of JSV Job Specifications.....	142
3.8.6.2	Using JSVs for Integrating Univa Grid Engine With Other Facilities.....	148
3.8.7	Enabling and Disabling Core Binding.....	148
3.8.7.1	Example: Enabling Core Binding on Host host1.....	149
3.8.8	The Univa Grid Engine Hadoop Integration.....	149
3.8.8.1	Brief Introduction to Apache Hadoop.....	149
3.8.8.2	Benefits of Using Hadoop with Univa Grid Engine.....	149
3.8.8.3	How Hadoop is Integrated with Univa Grid Engine.....	150
3.8.8.4	Installing the Hadoop Integration.....	150
3.8.8.4.1	Installation Prerequisites.....	151
3.8.8.4.2	Installing the Hadoop Integration Package.....	151
3.8.8.4.3	Verifying the Hadoop Integration.....	152
3.8.8.4.4	Troubleshooting the Hadoop Integration.....	153
3.9	Ensuring High Availability.....	153
3.9.1	Prerequisites.....	154
3.9.2	Installation.....	154
3.9.3	Testing sge_shadowd Takeover.....	154
3.9.4	Migrating the Master Host Back After a Takeover.....	155
3.9.5	Tuning the sge_shadowd.....	155
3.9.6	Troubleshooting.....	156
3.9.6.1	How do I know which host is currently running sge_qmaster?.....	156
3.9.6.2	Where do I find run time information of running shadow daemons?.....	156
3.9.6.3	Startup of sge_qmaster on a shadow host failed. Where do I find information for analyzing the problem?.....	156
3.10	Utilizing Calendar Schedules.....	156
3.10.1	Commands to Configure Calendars.....	156
3.10.2	Calendars Configuration Attributes.....	157
3.10.3	Examples to Illustrate the use of Calendars.....	159
3.11	Setting Up Nodes for Exclusive Use.....	159
3.12	Deviating from a Standard Installation.....	160
3.12.1	Utilizing Cells.....	160

Table of Contents

3 Special Activities

<u>3.12.2 Using Path Aliasing.....</u>	161
<u>3.13 Using CUDA Load Sensor.....</u>	162
<u>3.13.1 Building the CUDA load sensor.....</u>	162
<u>3.13.1.1 Example Makefile for building the CUDA load sensor.....</u>	163
<u>3.13.2 Installing the load sensor.....</u>	163
<u>3.13.2.1 Add "cuda" complex value.....</u>	163
<u>3.13.2.2 Assign "cuda" complex value to CUDA-enabled nodes.....</u>	164
<u>3.13.2.3 Install the load sensor.....</u>	164
<u>3.13.2.4 Installation Test.....</u>	164
<u>3.13.3 Using the CUDA load sensor.....</u>	164
<u>3.14 Special Tools.....</u>	165
<u>3.14.1 The Loadcheck Utility.....</u>	165
<u>3.14.2 Utilities for BDB spooling.....</u>	166

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/UGE800` 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/UGE800/default/common/setting.sh
```

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

```
> source /opt/UGE800/default/common/setting.csh
```

TABLE: Environment Variables Set by the `setting` Script

Environment Variable	Description
<code>\$SGE_ROOT</code>	The absolute path to the Univa Grid Engine product installation.
<code>\$ARCH</code>	The Univa Grid Engine architecture string. It identifies the OS and in some cases the processor architecture.
<code>\$SGE_CELL</code>	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 <code>\$SGE_ROOT</code>).
<code>\$SGE_CLUSTER_NAME</code>	The system wide unique name of the Univa Grid Engine cluster.
<code>\$SGE_QMASTER_PORT</code>	Network port where the master daemon is listening.
<code>\$SGE_EXECD_PORT</code>	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.
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.

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

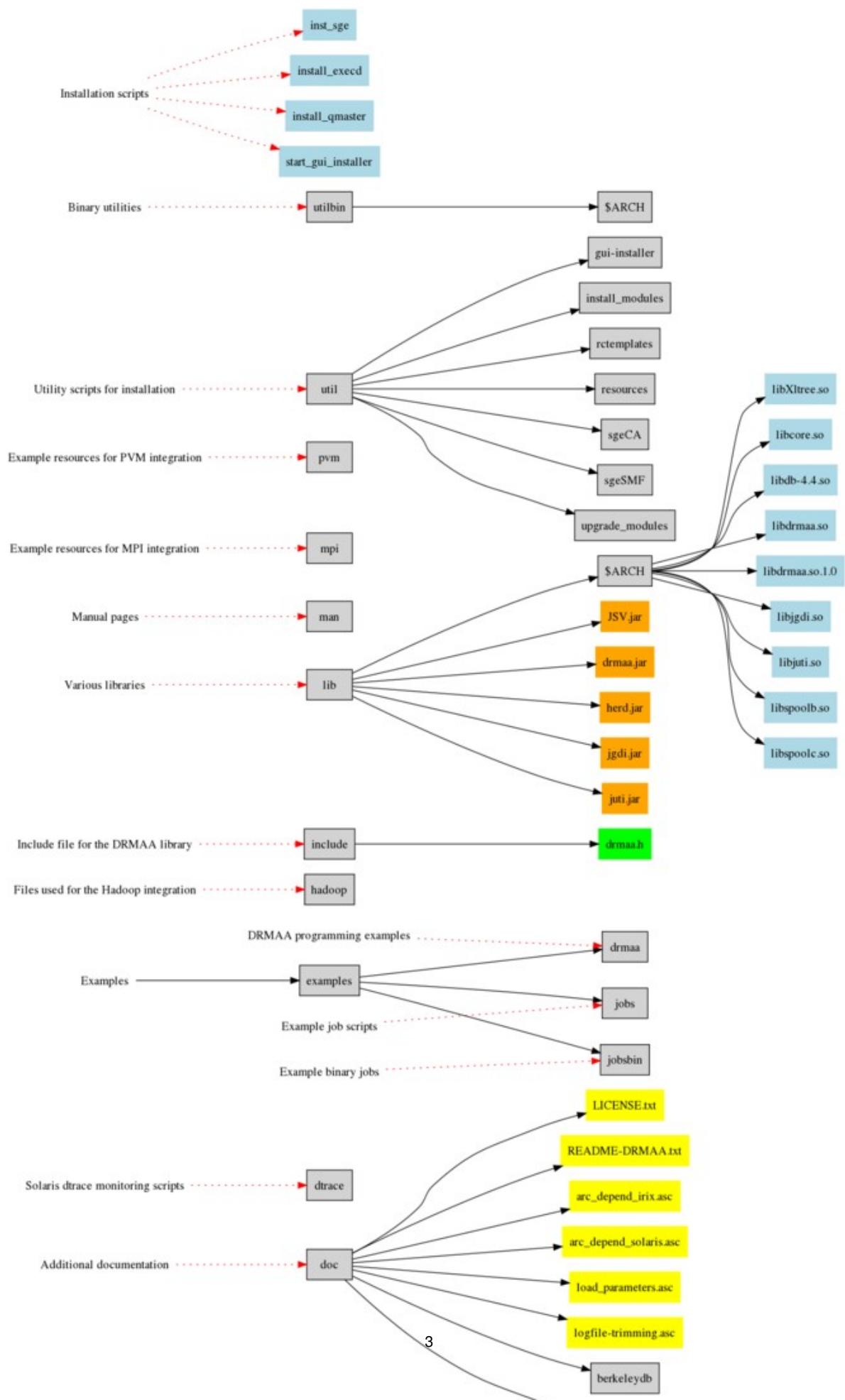




FIGURE 1: Overview of the \$SGE_ROOT Directory Structure

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.

TABLE: Overview of Main Configuration Files

Configuration File	Description
accounting	Contains accounting information about past jobs. The <code>qacct</code> client reads this data.
bootstrap	Contains information about spooling and multithreading for the <code>qmaster</code> .
configuration	Contains the current global cluster configuration, which can be modified by the <code>qconf -mconf</code> command.
qtask	The <code>qtask</code> configuration file (see man page <code>qtask</code>).
schedd_runlog	Contains information about a scheduling run, when it is monitored (see <code>qconf -tsm</code>).
cluster_name	Contains the unique cluster name.
sched_configuration	Contains the current scheduler configuration, which can be modified by the <code>qconf -tsm</code> 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 <code>qconf -mconf <hostname></code> command.

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. 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.

The following graphics illustrate the structure of the `qmaster` spooling directory (Figure 2) and the execution daemon spooling directory (Figure 3):

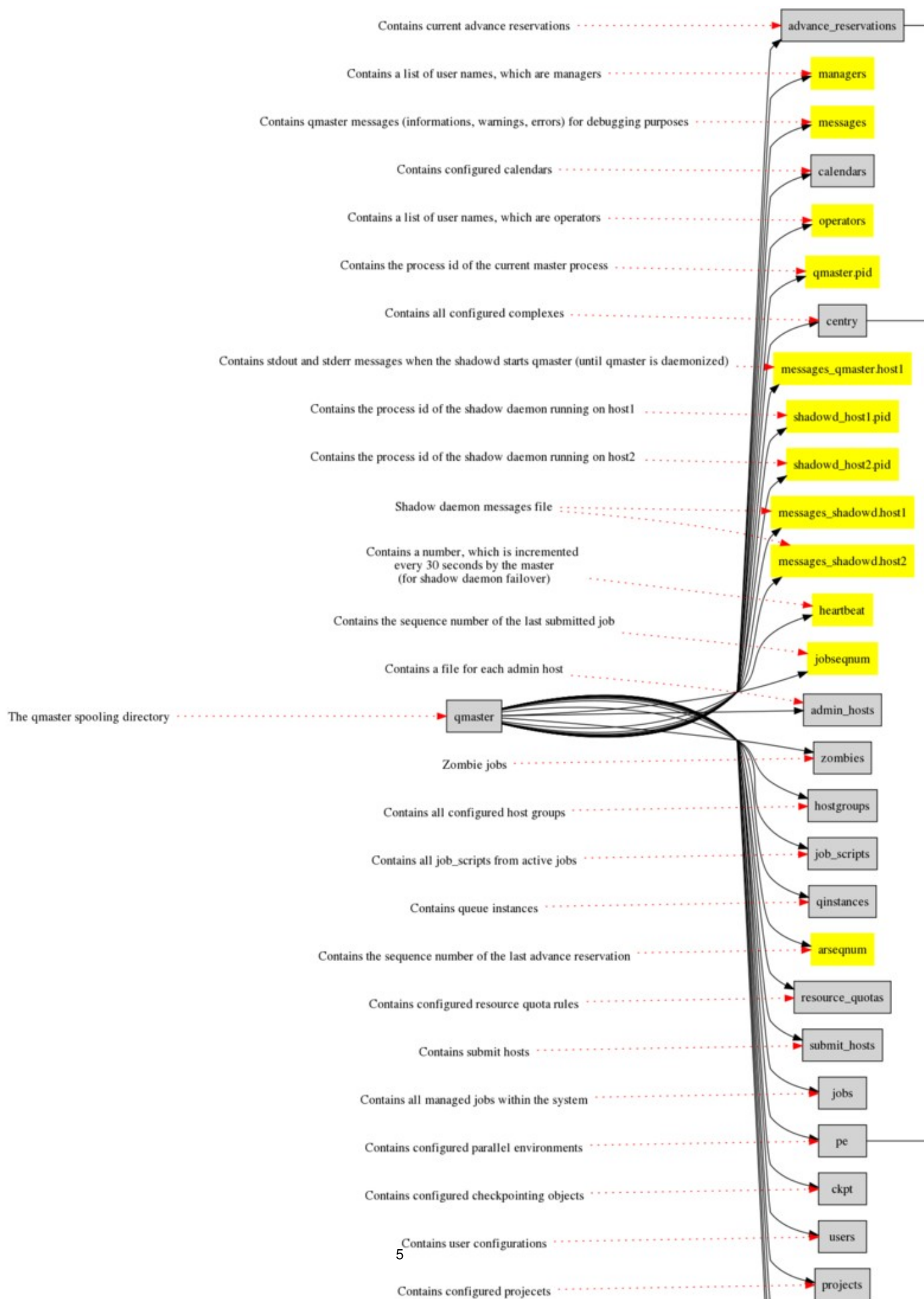




FIGURE 2: Overview of the qmaster Spooling Directory

[File:Execdspool.png](#)

FIGURE 3: Overview of the Execution Host Spooling 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.

1.1.2.1 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
13	0.50500	sleep	daniel	r	05/24/2011 09:57:07	all.q@host2
14	0.50500	sleep	daniel	r	05/24/2011 09:57:07	all.q@host3
15	0.50500	sleep	daniel	r	05/24/2011 09:57:07	all.q@host1
15	0.50500	sleep	daniel	r	05/24/2011 09:57:07	all.q@host1
15	0.50500	sleep	daniel	r	05/24/2011 09:57:07	all.q@host1
15	0.50500	sleep	daniel	r	05/24/2011 09:57:07	all.q@host1
15	0.50500	sleep	daniel	r	05/24/2011 09:57:07	all.q@host1
15	0.50500	sleep	daniel	r	05/24/2011 09:57:07	all.q@host1
12	0.60500	env	daniel	qw	05/24/2011 09:56:45	

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).

1.1.2.2 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
-----
   16 0.00000 sleep      daniel        hqw   05/24/2011 10:33:17
```

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
-----
   16 0.00000 sleep      daniel        qw    05/24/2011 10:33:17
```

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

```
> qhold 16
modified hold of job 16
```

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

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

TABLE: Overview of Job 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.

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.

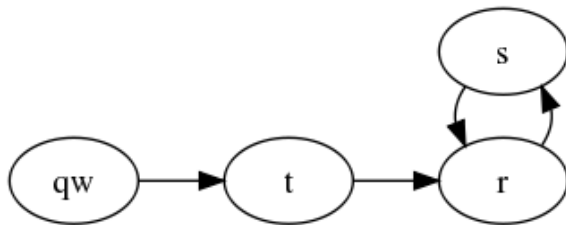


FIGURE 4: Simple Job State Transition

1.1.2.3 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
```

1.1.2.4 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
-----
```

- **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

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.

TABLE: Additional Standard Load Values

State	Description
<code>arch</code>	The architecture string (usually contains the OS and optionally the ISA).
<code>num_proc</code>	The number of detected processing units.
<code>mem_total</code>	The total amount of installed memory.
<code>swap_total</code>	The total amount of installed swap space.
<code>virtual_total</code>	Total amount of virtual memory (memory + swap space).
<code>load_avg</code>	Same as <code>load_medium</code> .
<code>load_short</code>	Average load value in the last minute (time interval may differ on OS; source on Linux is <code>/proc/loadavg</code>).
<code>load_medium</code>	Average load value in the last 5 minutes (time interval may differ on OS; source on Linux is <code>/proc/loadavg</code>).
<code>load_long</code>	Average load value in the last 15 minutes (time interval may differ on OS; source on Linux is <code>/proc/loadavg</code>).
<code>mem_free</code>	The amount of unused memory.
<code>swap_free</code>	The amount of unused swap space.
<code>virtual_free</code>	The amount of unused virtual memory.
<code>mem_used</code>	The amount of occupied memory.
<code>swap_used</code>	The amount of occupied swap space.
<code>virtual_used</code>	The amount of occupied virtual memory.
<code>cpu</code>	Current amount of CPU usage.
<code>m_topology</code>	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 the number of processors (num_proc).
np_load_medium	Medium load average divided by the the number of processors (num_proc).
np_load_long	Long load average divided by the the number of processors (num_proc).

1.1.2.5 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    1x-amd64
all.q@host2                             BIPC  0/0/10          0.08    1x-amd64
all.q@host3                             BIPC  0/0/10          0.01    1x-amd64
```

1.1.2.6 Understanding the Various Queue States

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

TABLE: Queue States

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 (<code>qmod -d</code>)
D	Automatically disabled
E	Error state

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 [Common_Tasks#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/csh
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
user_lists   NONE
xuser_lists  NONE
start_proc_args  NONE
stop_proc_args  NONE
allocation_rule $round_robin
control_slaves  TRUE
job_is_first_task FALSE
urgency_slots  min
accounting_summary TRUE
```

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 [Common_Tasks#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 -shgrp1
@allhosts
```

1.2.5 Default Complex Attributes

Many pre-defined *complex attributes* are available.

See [Common Tasks#Configuring Complexes](#) for additional information.

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

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

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 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.

1.4.2.1 Accounting

1.4.2.1.1 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](#).

TABLE: Information Available from view_accounting

Attribute	Type	Description
job_number	integer	the job id
task_number	integer	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.
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	integer	the job run time in seconds
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

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
```

1.4.2.2 Job related Information

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

1.4.2.2.1 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.

TABLE: Information Available from the Job Log

Attribute	Type	Description
job_number	integer	the job id
task_number	integer	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 <code>qsub -A</code> 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

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 12:02:17	pending		joga	new job
59708	2011-05-24 12:02:35	sent	t	master	sent to execd
59708	2011-05-24 12:02:36	delivered	r	master	job received by ex
59708	2011-05-24 12:02:44	suspended	r	joga	
59708	2011-05-24 12:03:01	unsuspended	r	joga	
59708	2011-05-24 12:03:35	finished	r	execution daemon	job exited
59708	2011-05-24 12:03:35	finished	r	master	job waits for sche
59708	2011-05-24 12:03:35	deleted	T	scheduler	job deleted by sch

1.4.2.2.2 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.

TABLE: Information Available from view_job_times

Attribute	Type	Description
job_number	integer	the job id
task_number	integer	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		the job submission time

	timestamp without time zone	
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

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_number	task_number	submission_time	wait_time	start_time	
4732	34	2011-05-23 14:32:43	00:03:07	2011-05-23 14:35:50	2011-
4695	-1	2011-05-23 14:28:49	00:03:08	2011-05-23 14:31:57	2011-
4732	35	2011-05-23 14:32:43	00:03:09	2011-05-23 14:35:52	2011-
4732	36	2011-05-23 14:32:43	00:03:17	2011-05-23 14:36:00	2011-
4732	37	2011-05-23 14:32:43	00:03:20	2011-05-23 14:36:03	2011-
4732	38	2011-05-23 14:32:43	00:03:28	2011-05-23 14:36:11	2011-

1.4.2.2.3 view_jobs_completed

The view_jobs_completed shows the number of jobs finished per hour.

TABLE: Information Available from view_jobs_completed

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

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

1.4.2.3 Advance Reservation Data

1.4.2.3.1 view_ar_attribute

The view_ar_attribute shows the basic attributes of an advance reservation.

TABLE: Information Available from view_ar_attribute

Attribute	Type	Description
ar_number	integer	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 qsub -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

Example:

```
SELECT * FROM view_ar_attribute;
```

ar_number	owner	submission_time	name	account	start_time	end_time
1	joga	2011-05-24 09:59:48		sge	2011-05-24 10:30:00	2011-05-24 11:

1.4.2.3.2 view_ar_log

TABLE: Information Available from view_ar_log

Attribute	Type	Description
ar_number	integer	the ar number
time	timestamp without time zone	time when the event logged occurred
event	text	type of the event
state	text	the state of the advance reservation
message	text	a message describing the event

Example:

```
SELECT * FROM view_ar_log;
```

ar_number	time	event	state	message
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

1.4.2.3.3 view_ar_resource_usage

TABLE: Information Available from view_ar_resource_usage

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

Example:

```
SELECT * FROM view_ar_resource_usage;
```

ar_number	variable	value
1	arch	sol-amd64

1.4.2.3.4 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.

TABLE: Information Available from view_ar_time_usage

Attribute	Type	Description
ar_id	numeric(38,0)	the ar number
job_duration	interval	actual usage of the reserved resources by jobs

ar_duration	interval	duration (time interval) of the advance reservation
-------------	----------	---

Example:

```
SELECT * FROM view_ar_time_usage;
```

```

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

```

1.4.2.3.5 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.

TABLE: Information Available from view_ar_usage

Attribute	Type	Description
ar_number	integer	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

Example:

```
SELECT * FROM view_ar_usage;
```

```

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

```

1.4.2.4 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.

1.4.2.4.1 view_department_values

TABLE: Information Available from 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)

1.4.2.4.2 view_group_values

TABLE: Information Available from 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)
--	--	--

1.4.2.4.3 view_host_values

TABLE: Information Available from 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)

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
...			

1.4.2.4.4 view_project_values

TABLE: Information Available from 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)

1.4.2.4.5 view_queue_values

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

TABLE: Information Available from view_queue_values

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
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)

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

1.4.2.4.6 view_user_values

TABLE: Information Available from 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
num_config	double precision	configured capacity of the value (for consumables)

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

1.4.2.5 Statistics

1.4.2.5.1 view_statistic

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

TABLE: Information Available from view_statistic

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

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.

1.4.3.1 Job Data and Accounting

1.4.3.1.1 sge_job

TABLE: Information Available from sge_job

Attribute	Type	Description
j_id	numeric(38,0) not null	internal sequential id
j_job_number	integer	the job number
j_task_number	integer	the array task number, -1 for sequential jobs
j_pe_taskid	text	the id of tasks of tightly integrated parallel jobs, -1
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
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

1.4.3.1.2 sge_job_log

TABLE: Information Available from 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

1.4.3.1.3 sge_job_request

TABLE: Information Available from 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

1.4.3.1.4 sge_job_usage

Holds job accounting data.

See also man page accounting.5 for details.

TABLE: Information Available from sge_job_usage

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	integer	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		
ju_ru_issmrss		
ju_ru_idrss		
ju_ru_isrss		
ju_ru_minflt		
ju_ru_majflt		
ju_ru_nswap		
ju_ru_inblock		
ju_ru_outblock		
ju_ru_msgsnd		

ju_ru_msgrcv		
ju_ru_nsignals		
ju_ru_nvcsw		
ju_ru_nivcsw		
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_maxvmem	double precision	the maximum vmem size in bytes.
ju_ar_parent	numeric(38,0) default 0	reference to sge_ar.ar_id

1.4.3.2 Advance Reservation Data

1.4.3.2.1 sge_ar

TABLE: Information Available from sge_ar

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

1.4.3.2.2 sge_ar_attribute

TABLE: Information Available from 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 qsub -A option
ara_start_time	timestamp without time zone	start time of the AR
ara_end_time		end time of the AR

	timestamp without time zone	
ara_granted_pe	text	if a parallel environment was requested at AR submission time, the name of the granted parallel environment

1.4.3.2.3 sge_ar_log

TABLE: Information Available from 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

1.4.3.2.4 sge_ar_resource_usage

TABLE: Information Available from 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

1.4.3.2.5 sge_ar_usage

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

TABLE: Information Available from sge_ar_usage

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

1.4.3.3 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

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.

1.4.3.3.1 sge_department

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

TABLE: Information Available from sge_department

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

1.4.3.3.2 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.

TABLE: Information Available from sge_group

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

1.4.3.3.3 sge_host

The sge_host table holds the names of all execution hosts.

TABLE: Information Available from sge_host

Attribute	Type	Description
-----------	------	-------------

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

1.4.3.3.4 sge_project

The sge_project table holds project names.

TABLE: Information Available from sge_project

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

1.4.3.3.5 sge_queue

The sge_queue table holds one record per queue instance.

TABLE: Information Available from sge_queue

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

1.4.3.3.6 sge_user

TABLE: Information Available from sge_user

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

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

1.4.3.3.7 sge_department_values

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

TABLE: 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		
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

1.4.3.3.8 sge_group_values

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

TABLE: Information Available from sge_group_values

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		
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

1.4.3.3.9 sge_host_values

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

TABLE: Information Available from sge_host_values

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		
hv_variable	text	variable name
hv_svalue	text	variable value for string variables
hv_dvalue	double precision	variable value for numeric variables
hv_dconfig	double precision	configured capacity for consumables

1.4.3.3.10 sge_project_values

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

TABLE: Information Available from sge_project_values

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		
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

1.4.3.3.11 sge_queue_values

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

TABLE: Information Available from sge_queue_values

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		
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

1.4.3.3.12 sge_user_values

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

TABLE: Information Available from sge_user_values

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		
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

1.4.3.4 Sharetree Usage

1.4.3.4.1 sge_share_log

TABLE: Information Available from 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

Index:

```
"sge_share_log_pkey" PRIMARY KEY, btree (sl_id)
```

1.4.3.5 Statistics

1.4.3.5.1 sge_statistic

TABLE: Information Available from sge_statistic

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

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 (s_id) REFERENCES "sge_statistic" (s_id)
```

1.4.3.5.2 sge_statistic_values

TABLE: Information Available from 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	

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)
```

1.4.3.6 dbwriter Internal Data

1.4.3.6.1 sge_checkpoint

TABLE: Information Available from sge_checkpoint

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

Indexe:

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

1.4.3.6.2 sge_version

TABLE: Information Available from sge_version

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

Indexe:

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

Go back to the [Univa Grid Engine Documentation](#) main page.

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 downtimes 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 it to leave the jobs running until they end themselves. The following examples describe scenarios that might help to find 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 "*@<hostname>"
```

- 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 <hostname>
```

- 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 <jsv_path>
...
```

- `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 it to use the backup/restore functionality to set up a second, identical cluster. This is described in the section [Updating with Two Separate Clusters on the Same Resource Pool](#) in the Installation Guide. If doing that upgrade and not disabling the functionality of the old cluster, then 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.

1. Disable all queues in the new cluster.

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

2. 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.

3. Continue with the previous step as long as there are enabled compute resources in the old cluster.
4. 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 for adding new execution hosts 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 [Execution Host Installation](#) of the Installation Guide.

2. The current system is CSP enabled or there are already Windows hosts in the cluster, and the goal is to add an execution host running a Unix-based or Windows 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 Interactive Installation of the Installation Guide, and then do the regular execution host installation described in 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.

1. Enable Windows Domain accounts.
 - ◆ This is done by adding `enable_windomacc=true` to the list of `execd_params`:

```
# qconf -mconf
...
execd_params enable_windomacc=true, ...
```
2. Add Windows Administrator as a manager.
 - ◆ The windows *Administrator* user must also be an administrator for Univa Grid Engine.

```
# qconf -am <windows_admin_name>
```
 - ◆ Note that the name of the *Administrator* account might be different on any given windows system.
3. Enable the security framework.

```
# $SGE_ROOT/util/sgeCA/sge_ca -init -days 365
```
4. **CSP-only** Create user certificates.
 - ◆ If the master was installed with security enhancements (CSP) enabled, then create certificates for the Windows Administrator.

```
# $SGE_ROOT/util/sgeCA/sge_ca -user <windows_admin_name>
```
5. Make the new windows host an admin host.

```
# qconf -ah <new_windows_hosts>
```
6. Follow the steps:
 - ◆ *Transfer Certificate Files and Private Keys (Manually)* in section Interactive Installation of the Installation Guide.
 - ◆ Then do a regular execution host installation on the new Windows host as described in Execution Host Installation.

2.1.4 Generate/Renew Certificates and Private Keys for Users

Follow these steps for the easiest way to create required certificates and private keys for those users that want to access a CSP secured Univa Grid Engine system:

1. 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
```

2. 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>
```

3. 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
...
```

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

Renew existing certificates:

1. 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 DAEMON_DAYS = 365
# extend the validity of the user certificate by
set USER_DAYS = 365
```

2. Renew the certificates.

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

3. 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}/ce
```

- To check the issuer:

```
# $SGE_ROOT/utilbin/${SGE_ARCH}/opensslx509 -issuer -in ~/.sge/port/${SGE_QMASTER_PORT}/${SGE_
```

- To show validity:

```
# $SGE_ROOT/utilbin/${SGE_ARCH}/opensslx509 -dates -in ~/.sge/port/${SGE_QMASTER_PORT}/${SGE_C
```

- To show the fingerprint:

```
# $SGE_ROOT/utilbin/${SGE_ARCH}/opensslx509 -fingerprint -in ~/.sge/port/${SGE_QMASTER_PORT}/${
```

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 that can later be used to restore the configuration. The backup saves configuration objects like queues, parallel environments, global/local cluster configuration. Then it saves important files located below `$SGE_ROOT`, but it does *not* save information about pending or running jobs. Due to that, the jobs will not be restored during the restore operation.

2.1.5.1 Creating a Manual Backup

To perform a backup manually, do the following steps.

1. Prepare the backup.

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

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

- ◆ Start the backup process.

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

2. 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 backupfiles?

◆ 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 tarpackage with your files? (y/n)

◆ Enter the backup file name.

Please enter a filename for your backupfile.

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

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

3. Verify.

- ◆ Verify that the backup file was created.

2.1.5.2 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.

2.1.5.3 Restoring from a Backup

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

1. Prepare to restore.

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

```
# source $SGE_ROOT/default/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.
```

2. 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 backupfile 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
```

3. 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.**

2.2 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.
- It is also recommended that the user id and primary group id be the same on all hosts.
- 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.
- In CSP enabled system, it is also necessary to take the steps described in section Generate/Renew Certificate and Private Keys so that each user has access to a certificate and private keyfile to be able to use commands to submit jobs.
- Users need to be able to read the files in `$SGE_ROOT/$SGE_CELL/default/common`, and it is recommended to have full access to the directory referenced by `$TMPDIR`.

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

- **Advanced Reservations**
Users are not allowed to create advanced 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 Administrators

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

The admin user that is defined during the installation process of the Univa Grid Engine software is automatically an administrator. 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 administrators, the default admin user will also have file access to central configuration files.

Commands to Manage Administrators

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

- `qconf -sm`
Displays the list of all uses with administrator rights.
- `qconf -am <username>`
Makes the specified user an administrator.
- `qconf -dm <username>`
Deletes the specified user from the list of administrators.

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 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.

2.2.4.1 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.

2.2.4.2 Configuration Parameters of Access Lists

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

TABLE: Access List Configuration Attributes

Attribute	Value Specification
<code>name</code>	The name of the access list.
<code>type</code>	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.
<code>oticket</code>	The number of override tickets currently assigned to the department.
<code>fshare</code>	The current functional shares of the department.
<code>entries</code>	<p>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.</p> <p>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 <code>defaultdepartment</code> is assigned, if existing.</p>

2.2.5 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.

2.2.5.1 Commands to Add, Modify Delete Projects

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

- `qconf -apj`
Adds a new project.
- `qconf -Aprj <filename>`
Adds a new project that is defined in the specified file.
- `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>`
Modifies the project. New object configuration is taken from the specified file.
- `qconf -sprj`
Shows the current configuration of the project.
- `qconf -sprjl`
Shows all existing projects of an Univa Grid Engine cluster.

2.2.5.2 Configuration Parameters of Projects

Each project object supports the following set of configuration attributes:

TABLE: Project Configuration Attributes

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

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 -sconf1 <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)`.

TABLE: Project Configuration Attributes

Attribute	Value Specification
<code>execd_spool_dir</code>	The execution daemon spool directory.
<code>mailer</code>	Absolute path to the mail delivery agent.
<code>load_sensor</code>	A comma separated list of executables to be started by execution hosts to retrieve site configurable load information.
<code>prolog</code> <code>epilog</code>	Absolute path to executables that will be executed before/after a Univa Grid Engine job.
<code>shell_start_mode</code>	Defines the mechanisms which are used to invoke the job scripts on execution hosts.
<code>min_uid</code> <code>min_gid</code>	Defines the lower bound on user/group IDs that may use the cluster.
<code>user_lists</code> <code>xuser_lists</code>	User access lists that define who is allowed access to the cluster.
<code>administrator_mail</code>	List of mail addresses that will be used to send problem reports.
<code>project</code> <code>xproject</code>	Defines which projects are granted access and where access will be denied.
<code>load_report_time</code>	The system load of execution hosts is periodically reported to the master host. This parameter defines the time interval between load reports.
<code>reschedule_unkown</code>	Determines whether jobs on execution hosts in an unknown state are rescheduled and thus sent to other hosts.
<code>max_unheard</code>	If the master host could not be contacted or was not contacted by the execution daemon of a host for <code>max_unheard</code> seconds, all queues residing on that particular host are set to status unknown.
<code>loglevel</code>	Defines the detail level for log messages.
<code>max_aj_instances</code>	

	This parameter defines the maximum number of array tasks to be scheduled to run simultaneously per array job.
<code>max_aj_tasks</code>	Defines the maximum number of tasks allowed for array jobs. If this limit is exceeded, then the job will be rejected during submission.
<code>max_u_jobs</code>	The number of active jobs each user can have in the system simultaneously.
<code>max_jobs</code>	The number of active jobs in the system.
<code>max_advance_reservations</code>	The maximum number of active advance reservations allowed in Univa Grid Engine.
<code>enforce_project</code>	When set to true, users are required to request a project during submission of a job.
<code>enforce_user</code>	When set to <code>true</code> , users must exist within the Univa Grid Engine system before they can submit jobs. <code>auto</code> means that the user will be automatically created during the submission of the job.
<code>auto_user_delete_time</code> <code>auto_user_default_project</code> <code>auto_user_fshare</code> <code>auto_user_oticket</code>	Defines different aspects for automatically created users.
<code>gid_range</code>	Comma separated list of range expressions specifying additional group IDs that might be used by execution daemons to tag jobs.

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 load 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 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
queue_sort_method         load
job_load_adjustments      np_load_avg=0.15
load_adjustment_decay_time 0:7:30
load_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          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
```

The scheduler parameters are explained in the table below:

TABLE: Scheduler Configuration Attributes

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.

<code>maxujobs</code>	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.
<code>queue_sort_method</code>	Defines in which way the queue instance list is sorted (this implies the order of hosts) and matched against the job list. The default parameter is <code>load</code> (i.e. the <code>load_formula</code> is taken). Alternatively, <code>seqno</code> determines that the order is based on the sequence numbers of the queue instances, which is defined in the queue configuration.
<code>job_load_adjustment</code>	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).
<code>load_adjustment_decay_time</code>	The load adjustment is scaled linearly. This means right after dispatching the job, the <code>job_load_adjustment</code> adjusts the load value of the resources with 100% influence. After a while the influence is reduced linearly until <code>load_adjustment_decay_time</code> is reached.
<code>load_formula</code>	When the <code>queue_sort_method</code> is set to <code>load</code> , this formula describes how the load is derived. The default is <code>np_load_avg</code> the normalized average load.
<code>sched_jobinfo</code>	If set to <code>true</code> , additional scheduling information can be seen in the <code>qstat -j</code> output. The default value is <code>false</code> , because it impacts the overall scheduler performance in bigger clusters.
<code>flush_submit_sec</code>	If unequal to 0, it defines an additional scheduler run that is performed the specified number of seconds after a job was submitted.
<code>flush_finish_sec</code>	If unequal to 0, it defines an additional scheduler run that is performed the specified number of seconds after a job finishes.
<code>params</code>	Additional parameters for the Univa Grid Engine scheduler: <ul style="list-style-type: none"> • <code>DURATION_OFFSET</code> : assumed offset between run-time of a job and the run-time from scheduler perspective • <code>PROFILE</code> : 1 = turning run-time profiling on

	<ul style="list-style-type: none"> • 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

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 the 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**.

2.5.1.1 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:

TABLE: Local Cluster Configuration

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

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 `sgc_conf` and in the section [Understanding and Modifying the Cluster Configuration](#).

2.5.1.2 Execution Host Configuration

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

TABLE: Execution Host Configuration

Attribute	Value Specification
<code>qconf -ae</code> [<template>]	Adds a new execution host configuration, optionally based on a configuration template.
<code>qconf -Ae</code> <filelist>	Adds an execution host configuration based on a file.
<code>qconf -de</code> <hostlist>	Deletes execution host configuration based on the given host list.
<code>qconf -me</code> <hostname>	Modifies the execution host configuration of the host given by the hostname.
<code>qconf -Me</code> <filename>	Modifies an execution host given based on a configuration file.
<code>qconf -se</code> <hostname>	Shows the execution host configuration of the given host.

The following is an example of an execution host configuration:

```

> qconf -se macsuse
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](#).
- 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 `xusers_lists`), have access to the host.
 - ◆ All users in the access lists of `xusers_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.

2.5.1.3 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.

TABLE: Admin Host Configuration

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

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

TABLE: Submission Host Configuration

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

2.5.1.4 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.

TABLE: Host Group Configuration

Attribute	Value Specification
<code>qconf -ahgrp <group></code>	Adds a new host group entry with the name <group> and opens an editor for editing.
<code>qconf -Ahgrp <filename></code>	Adds a new host group entry with the configuration based on the file <filename>.
<code>qconf -dhgrp <group></code>	Deletes the host group with the name <group>.
<code>qconf -mhgrp <group></code>	Modifies the host group <group> in an interactive editor session.

<code>qconf -Mhgrp <filename></code>	Modifies a host group based on a configuration file <filename>.
<code>qconf -shgrp <group></code>	Shows the configuration of the host-group <group>.
<code>qconf -shgrp_tree <group></code>	Shows the host-group <group> with sub-groups in a tree structure.
<code>qconf -shgrp_resolved <group></code>	Shows the host-group with an resolved host-list.
<code>qconf -shgrp1</code>	Shows a list of all host-groups.

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.

2.5.1.5 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.

TABLE: Queue Configuration Commands

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

2.5.2.1 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)
daniel@tanqueray 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/csh
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
daniel@tanqueray removed "new.q" from cluster queue list
```

2.5.2.2 Queue Configuration Attributes

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

2.5.2.2.1 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:

TABLE: Queue Resource Limits

Attribute	Value Specification
<code>h_rt</code>	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.
<code>s_rt</code>	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 <code>notify</code> parameter), the job is killed.
<code>h_cpu</code>	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.
<code>s_cpu</code>	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.

<code>h_vmem</code>	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.
<code>s_vmem</code>	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.
<code>h_fsize, s_fsize, h_data, s_data, h_stack, s_stack, h_core, s_core, h_rss, s_rss</code>	These limits have the semantic of the <code>setrlimit</code> system call of the underlying operating system.

2.5.2.2.2 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.

2.5.2.2.3 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 `sgc_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.

2.5.2.2.4 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.

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.

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`).

2.5.2.2.5 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.

2.5.2.2.6 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](#).

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 [Implementing Pre-emption Logic](#).

2.5.2.2.7 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).

2.5.2.2.8 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.

2.5.3.1 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 (=).

2.5.3.1.1 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 table:

TABLE: *Complex Configuration Commands*

Attribute	Value Specification
<code>qconf -sc</code>	Shows a list of all configured complex entries (the complex table).
<code>qconf -mc</code>	Opens the editor configured in the <code>\$EDITOR</code> 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.
<code>qconf -Mc <filename></code>	Reads the given file in as new complex configuration.

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

TABLE: *Complex Configuration Attributes*

Attribute	Value Specification
<code>name</code>	The unique name of the complex.
<code>shortcut</code>	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.
<code>type</code>	The type of the complex variable (used for internal compare functions and load scaling). Possible values are <code>INT</code> , <code>DOUBLE</code> , <code>TIME</code> , <code>MEMORY</code> , <code>BOOL</code> , <code>STRING</code> , <code>CSTRING</code> , <code>RESTRING</code> and <code>HOST</code> . See <code>man complex</code> for more detailed format descriptions. A <code>CSTRING</code> is a case insensitive string type.

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.
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 at ...).

After a default installation, the following complexes are available:

```
> qconf -sc
```

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

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

2.5.3.1.2 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 queueuname
...
complex_values  complexname=20,[host1=complexname=10],[host2=complexname=5]
```

2.5.3.1.3 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
queueuname                                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.

TABLE: Meaning of Different Prefixes from Complexes Shown by `qstat` and `qhost`

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
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)

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
```

2.5.3.2 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.

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.

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 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:

TABLE: Commands for Direct Object Modification

Attribute	Value Specification
<code>qconf -aattr obj_nm attr_nm val obj_id_list</code>	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 <code>obj_id_list</code>).
<code>qconf -Aattr obj_nm fname obj_id_list</code>	Same as above but the attribute name and attribute value is taken from a file given by the file name <code>fname</code> .
<code>qconf -dattr obj_nm attr_nm val obj_id_list</code>	Deletes an object attribute.
<code>qconf -Dattr obj_nm fname obj_id_list</code>	Deletes an object attribute by a given file.
<code>qconf -mattr obj_nm attr_nm val obj_id_list</code>	Modifies an object attribute.
<code>qconf -Mattr obj_nm fname obj_id_list</code>	Modifies an object attribute based on a given file.
<code>qconf -purge obj_nm3 attr_nm objectname</code>	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.
<code>qconf -rattr obj_nm attr_nm val obj_id_list</code>	Replaces an object attribute.
<code>qconf -Rattr obj_nm fname obj_id_list</code>	Replaces an object attribute based on a given file.

2.5.4.1 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
daniel@macsuse 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
daniel@macsuse 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
daniel@macsuse 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
daniel@macsuse modified "all.q" in cluster queue list
```

Now show the queue configuration again:

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

2.6 Monitoring and Modifying User Jobs

Refer to section [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 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 -t sm` 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/comon/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".

2.7.2 Location of Logfiles and Interpreting Them

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

TABLE: Daemon Log File Locations

Daemon	Log file
sge_qmasters	<sge_qmaster_spool_dir>/messages
sge_shadowd	<sge_qmaster_spool_dir>/messages_shadowd.<host>
sge_sgeexecd	<sge_execd_spool_dir>/messages
sge_shepherd	<sge_execd_spool_dir>/active_jobs/<job_dir>/trace
dbwriter	\$SGE_ROOT/\$SGE_CELL/common/spool/dbwriter/dbwriter.log

- <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 {{GEprefixLC}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 <host>").
- <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
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 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. 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://
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/dbwrit
23/05/2011 14:14:19|kailua|Writer$VacuumAnalyzeThread.run|I|Next vacuum analyze will be execute
23/05/2011 14:14:19|kailua|ngDBWriter$StatisticThread.run|I|Next statistic calculation will be
23/05/2011 14:15:19|kailua|er.file.FileParser.processFile|I|Renaming reporting to reporting.pr
23/05/2011 14:15:19|kailua|iter.file.FileParser.parseFile|W|0 lines marked as erroneous, these
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 its messages file - e.g. because of insufficient permissions, because the directory doesn't exist or isn't accessible, and so on - then it writes a panic file to the /tmp directory. The names of these panic files are related to the name of the module that wrote them, and the pid of the module is always appended to the file name. E.g. `execd_messages.<pid>` or `shepherd.<pid>`

2.7.3 Turning on Debugging Information

The debugging sections describe recommended debugging tools available in Univa Grid Engine, including scheduler profiling, logfiles and DTrace.

2.7.3.1 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(<timestamp>):*.

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, pass2: 0.000, c
04 PROF: job ticket calculation: init: 0.000 s, pass 0: 0.000 s, pass 1: 0.000, pass2: 0.000, c
05 PROF: normalizing job tickets took 0.010 s
06 PROF: create active job orders: 0.010 s
07 PROF: job-order calculation took 0.090 s
08 PROF: job sorting took 0.090 s
09 PROF: job dispatching took 0.000 s (20 fast, 0 fast_soft, 0 pe, 0 pe_soft, 0 res)
10 PROF: parallel matching          global          rqs          cqstatic          hstatic          qstatic
11 PROF: sequential matching        global          rqs          cqstatic          hstatic          qstatic
12 PROF: parallel matching          0              0              0              0              0
13 PROF: sequential matching        20             0              20             20             20
14 PROF: create pending job orders: 0.050 s
15 PROF: scheduled in 0.310 (u 0.220 + s 0.000 = 0.220): 20 sequential, 0 parallel, 11799 order
16          11775 J(qw), 20 J(r), 0 J(s), 0 J(h), 0 J(e), 0 J(x), 11795 J(all), 52 C, 1 ACL, 1 PE,
17          0 ST, 1 CKPT, 0 RU, 1 gMes, 0 jMes, 11799/4 pre-send, 0/0/0 pe-alg
18 PROF: send orders and cleanup took: 0.090 (u 0.080,s 0.000) s
19 PROF: schedd run took: 0.720 s (init: 0.000 s, copy: 0.200 s, run:0.490, free: 0.000 s, jobs

```

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 1* 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 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 = 6.050
03	PROF(1664087824): packing	: wc =	0.000s,	utime =	0.000s, stime = 0.000
04	PROF(1664087824): eventclient	: wc =	0.000s,	utime =	0.000s, stime = 0.000
05	PROF(1664087824): mirror	: wc =	0.020s,	utime =	0.000s, stime = 0.040
06	PROF(1664087824): gdi	: wc =	0.000s,	utime =	0.000s, stime = 0.000
07	PROF(1664087824): ht-resize	: wc =	0.000s,	utime =	0.000s, stime = 0.000
08	PROF(1664087824): scheduler	: wc =	0.910s,	utime =	0.340s, stime = 0.040
09	PROF(1664087824): pending ticket	: wc =	0.130s,	utime =	0.000s, stime = 0.000
10	PROF(1664087824): job sorting	: wc =	0.110s,	utime =	0.120s, stime = 0.020
11	PROF(1664087824): job dispatching:	wc =	0.000s,	utime =	0.000s, stime = 0.000
12	PROF(1664087824): send orders	: wc =	0.380s,	utime =	0.360s, stime = 0.050
13	PROF(1664087824): scheduler event:	wc =	0.180s,	utime =	0.110s, stime = 0.010
14	PROF(1664087824): copy lists	: wc =	1.270s,	utime =	0.260s, stime = 0.440
15	PROF(1664087824): total	: wc =	60.340s,	utime =	2.070s, stime = 6.790

2.7.3.2 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.

2.7.3.2.1 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
```

2.7.3.2.2 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:

<code>weight_priority</code>	1.000000
<code>weight_urgency</code>	0.100000
<code>weight_ticket</code>	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:

<code>job-ID</code>	<code>prior</code>	<code>name</code>
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.

2.7.3.3 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/default/common/settings.sh
# . $SGE_ROOT/util/dl.sh
# dl <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/common/sge_qstat" does not exist
04	3	17230	main	file "/Users/ernst/.sge_qstat" does not exist
05	4	17230	main	queues not needed
06	5	17230	main	sge_set_auth_info: username(uid) = user1(500), groupname = user1(500)
07	6	17230	main	----- selecting queues -----
08	7	17230	main	----- selecting jobs -----
09	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.

2.7.3.4 Using DTrace for Bottleneck Analysis

DTrace is a dynamic tracing framework available for different operating systems like FreeBSD, NetBSD, Mac OS X or Solaris. This application can be used by developers and administrators to retrieve helpful information about running processes. In combination with Univa Grid Engine, this Software can be used on the master machine to receive dynamic events with data about qmasters:

- spooling framework
- message processing
- scheduling
- synchronization between scheduler and other master threads
- communication framework
- locking infrastructure

The DTrace tool can be started with the `monitor.sh` script that is located in `$SGE_ROOT/dtrace` directory after the installation of the Univa Grid Engine system. The `monitor.sh` script needs to be executed as user root on the machine on which the

`sge_qmaster` process is running in order to receive dynamic events.

By default, the `monitor.sh` script prints one line of output for each statistics interval. The default interval is 15 seconds, but this can be changed with the `-interval` command line parameters. To see which function calls of spooling or request framework are called, including all parameters that are passed to those functions, it might be possible add the command line parameters `-spooling` and/or `-requests` when the monitoring script is started. If the cell name is different from *default*, then the cell name can be specified with the `-cell <cell_name>` parameter.

The following table describes the counters values of the statistics output.

TABLE: Columns Shown in DTrace Output

Topic	Name	Description
Spooling	#wrt	Number of qmaster write operations via <code>spool_write_object()</code> and <code>spool_delete_object()</code> . Almost any significant write operation goes through this function both in bdb/classic spooling.
	wrt	Total time all threads spend in <code>spool_write_object()</code> in micro seconds.
Synchronization	#snd	Number of event packages sent by qmaster to schedd. If that number goes down to zero over longer time there is something wrong and qmaster/schedd has gotten out of sync.
	#rcv	Number of event packages received by schedd from qmaster. If that number goes down to zero over longer time there is something wrong and qmaster/schedd has gotten out of sync.
Message Processing	#rep	Number of reports qmaster processed through <code>sge_c_report()</code> . Most data sent by execd's to qmaster comes as such a report (job/load/config report).
	#gdi	Number of GDI requests qmaster processed through <code>do_gdi_request()</code> . Almost anything sent from client commands arrives at qmaster as a GDI request, but execd's and scheduler also use GDI requests.
	#ack	Number of ACK messages qmaster processed through <code>do_c_ack()</code> . High numbers of ACK messages can be an indication of job signaling, but they are also used for other purposes.
Scheduling	#dsp	Number of calls to <code>dispatch_jobs()</code> in schedd. Each call to <code>dispatch_jobs()</code> can be seen as a scheduling run.
	dsp	Total time scheduler spent in all calls to <code>dispatch_jobs()</code> .
	#sad	Number of calls to <code>select_assign_debit()</code> . Each call to <code>select_assign_debit()</code> can be seen as a try of the scheduler

		to find an assignment or a reservation for a job.
Communication	#in++	Number of messages added into qmaster received messages buffer.
	#in--	Number of messages removed from qmaster received messages buffer. If more messages are added than removed during an interval, the total of messages not yet processed is about to grow.
	#out++	Number of messages added into qmaster send messages buffer.
	#out--	Number of messages removed from qmaster send messages buffer. If more messages are added than removed during an interval, the total of not yet messages not yet delivered is about to grow.
Locking	#lck0/#ulck0	Number of calls to sge_lock()/sge_unlock() for qmasters <i>global</i> lock. This lock must always be obtained, when qmaster-internal lists (job list, queue list, etc.) are accessed.
	#lck1/#ulck1	Number of calls to sge_lock()/sge_unlock() for qmasters <i>master_config</i> lock.

Go back to the [Univa Grid Engine Documentation](#) main page.

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

3.1.1.1 Setup 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_shadowd, 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.

3.1.1.2 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

[Common Tasks#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

3.1.3.1 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.

3.1.3.2 Switch off PDC

The PDC (Portable Data Collector) is a module in sge_execd which monitors resource usage of jobs (online usage) and enforces limits (e.g. wallclock limit) on jobs.

For jobs running only a few seconds, the online usage produced by PDC will probably never be used, and a short job will usually not run into wallclock or CPU limits.

In this case, consider switching off PDC. PDC can be switched off by modifying the global cluster configuration and adding `PDC_INTERVAL=NEVER` to the attribute `execd_params`.

3.1.4 Choosing Job Submission Options

Choosing the right options at job submission can have significant impact on

- submission time,
- scheduling time,
- job execution time.

The following submission options have negative impacts on scheduler performance. Try to avoid them if possible:

- Soft requests: `qsub -soft -l resource_list`.
- Wildcard parallel environments, e.g. `qsub -pe mpi* 4`.
- Resource reservation: `qsub -R yes`

For submission of a high number of similar jobs consider using array jobs, see [User Guide#Array Jobs](#).

For modeling job dependencies consider the use of qmake, see [User_Guide#qmake](#). Instead of submitting individual jobs with dependencies to Univa Grid Engine, the jobs can either

- be started as tasks within a single parallel job
- be submitted as individual jobs by qmake, but qmake limits the number of jobs (tasks) being submitted concurrently to Univa Grid Engine, thus keeping the total number of jobs in the cluster low.

[download this book as a PDF](#)

3.2 Tuning Univa Grid Engine for Large Parallel Applications

3.2.1 General Settings

3.2.1.1 Interactive Job Support

Large parallel jobs using the tight parallel job integration (see also [Parallel Environments](#)) start the tasks of the parallel job through the Univa Grid Engine command `qrsh -inherit`.

The session type to use for `qrsh -inherit` is configured in the cluster configuration, attributes `rsh_daemon` and `rsh_command`.

The choice of the session type can have significant impact on the performance of starting tasks, it is recommended to use the *builtin* session type. This is the default for a Univa Grid Engine installation.

See also [Interactive Jobs](#) for further information about the session types.

3.2.1.2 Accounting Summary

When running tightly integrated parallel jobs, the attribute *accounting_summary* of the parallel environment configuration defines if

- a single accounting record is generated for the whole job, or
- an individual accounting record is generated for the master task and for every slave task.

Generating individual accounting records for every task increases

- the communication between the `sge_qmaster` and the execution nodes,
- the spooling effort for `sge_qmaster` (it has to spool every individual slave task),
- the number of accounting records and the size of the accounting file.

In most cases the combined accounting for the whole job is what the user actually wants to see.

Setting *accounting_summary* to *TRUE* can significantly improve the overall performance of *sge_qmaster* and the execution nodes.

See also [Parallel Environments](#) for further information about configuration and use of parallel environments.

3.2.2 Tuning on the Execution Side

There are different options to shorten the time of job execution on the execution (*sge_execd*) side:

- tune *sge_execd* performance
- reduce impact of *sge_execd* on the execution host

3.2.2.1 Tuning *sge_execd*

Performance of *sge_execd*, especially at job / parallel task startup and end depends on the spooling setup of *sge_execd*.

When a job is delivered by *sge_qmaster* to *sge_execd* for start, or when a task of a tightly-integrated job is started by the master task, *sge_execd* spools the job data and optionally the job script, and it creates temporary files for startup and monitoring of the job (*active_jobs/<job_id>.<task_id>/...*).

When a job or task finishes, the job spooling and temporary files need to be deleted.

All spooling data is written in subdirectories of the *execd_spool_dir* configured in the global or local cluster configuration, see also [Understanding and Modifying the Cluster Configuration](#)

Performance of the spooling operations highly depends on the file system used for spooling.

It is recommended to do *sge_execd* spooling on a local file system.

For changing the *sge_execd* spool directory

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

3.2.2.2 Reducing Impact of *sge_execd* on the Execution Host

When a job is running on an execution host, the *sge_execd* is monitoring the processes belonging to the job. Job monitoring is done by the PDC - the Portable Data Collector. The PDC gathers per job usage information (cpu, memory and - on selected platforms - io) and is enforcing limits *h_cpu*, *s_cpu*, *h_vmem* and *s_vmem*.

The process monitoring can create noticeable load on the execution host, depending on the number of jobs running on the host and the number of processes (size of the process chain) initiated per job.

It can be configured in the (local or global) cluster configuration, attribute `execd_params`.

Setting the `execd_param PDC_INTERVAL` determines how often online usage of running jobs are gathered and limits are evaluated.

`PDC_INTERVAL` can be set to

- **NEVER**: No online usage is gathered. Switches off the PDC in `sge_execd`. Use this option only when
 - ◆ no sharetree is configured
 - ◆ online usage information of jobs isn't needed
 - ◆ ARCo (dbwriter) isn't used
- **PER_LOAD_REPORT**: `sge_execd` triggers a PDC run once per load report interval
- a time range specification (e.g. 0:0:2). PDC is triggered in the specified interval.

The default PDC interval is 1 second.

3.2.3 Job Related Tuning

Job performance can be influenced by submission options.

The following hints may improve the performance of large parallel jobs.

- Avoid exporting huge environments with the submit option `-V`. Instead export only selected environment variables, using the submit option `-v`.
- Make sure the job writes temporary information to the directory pointed to by the `TMPDIR` environment variable. Univa Grid Engine creates a temporary directory (`TMPDIR`) for every job and checks that it is deleted at job end. The location for the temporary directory is set in the queue configuration, see [Configuring Queues](#).
- If multiple jobs can run on a host, consider using job to core binding for running jobs, see [Jobs with Core Binding](#).

3.3 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 a 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 an Univa Grid Engine managed cluster.

3.3.1 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**).

```
> qghost
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:

```
> qghost | 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
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.

```
> qghost -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=5198.269531M, \
                        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:

```
> qghost -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
-----
      6 0.55500 sleep      daniel         r    04/14/2011 09:45:12 bq@macsuse
      7 0.55500 sleep      daniel         r    04/14/2011 09:45:12 bq@macsuse
      8 0.55500 sleep      daniel        qw    04/14/2011 09:44:25
      9 0.55500 sleep      daniel        qw    04/14/2011 09:44:25
```

More information about why certain jobs are not scheduled 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.

3.3.2 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.

3.3.2.1 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=5198.269531M, \
                        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`)

3.3.3 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
...
queue_sort_method          load
job_load_adjustments       np_load_avg=0.50
load_adjustment_decay_time 0:7:30
load_formula               np_load_avg
schedd_job_info            false
```

The **queue_sort_method** determines the order of the queue-instances when they are matched against the pending job list. Possible values for this attribute are **load** (default) and **seq_num**. The **load_formula** describes load type and is calculated in case of the queue sort method **load**.

3.3.3.1 Queue Sequence Number

When the **queue_sort_method** is changed to **seq_num**, 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.

3.3.3.1.1 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 `queue_sort_method` to `seq_no`.

```
> qconf -msconf
...
queue_sort_method      seq_no
...
```

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

3.3.3.1.2 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
4	0.55500	sleep	daniel	r	04/28/2011 14:31:36	a@host1
5	0.55500	sleep	daniel	r	04/28/2011 14:31:36	a@host2
6	0.55500	sleep	daniel	r	04/28/2011 14:31:36	a@host3
7	0.55500	sleep	daniel	r	04/28/2011 14:31:36	a@host4

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

3.3.3.1.3 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**.

3.4 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.

3.4.1 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 "myresourcequot
...
```

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 be 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 matches, 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
}
```

3.4.2 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 the them `host1` with the additional GPU processing facility.

```
> qhost
HOSTNAME                ARCH                NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
#-----
global                  -                  -    -    -        -        -        -
host1                   1x26-amd64         4  0.00  934.9M  134.1M  1004.0M  0.0
host2                   1x26-amd64         4  0.02   2.0G   430.9M   2.0G    0.0
host3                   1x26-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
```

queue	name	qtype	resv/used/tot.	load_avg	arch	states
all.q@host1	hc:GPU=1	BIPC	0/0/10	0.00	1x26-amd64	
all.q@host2		BIPC	0/0/10	0.00	1x26-amd64	
all.q@host3		BIPC	0/0/10	0.00	1x26-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
```

queue	name	qtype	resv/used/tot.	load_avg	arch	states
all.q@host1	hc:GPU=0	BIPC	0/1/10	0.00	1x26-amd64	
	4 0.55500 sleep	daniel	r	03/04/2011 10:17:21	1	
all.q@host2		BIPC	0/0/10	0.00	1x26-amd64	
all.q@host3		BIPC	0/0/10	0.00	1x26-amd64	

If a second GPU job is started, then let the scheduler run again (`-t sm`):

```
> qsub -b y -l GPU=1 sleep 100
Your job 5 ("sleep") has been submitted

> qconf -t sm
daniel@hostname triggers scheduler monitoring
```

```
> qstat
```

job-ID	prior	name	user	state	submit/start at	queue
4	0.55500	sleep	daniel	r	03/04/2011 10:17:21	all.q@host1
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
-----
      5 0.55500 sleep      daniel         r    03/04/2011 10:19:07 all.q@host1
```

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 define 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 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
```

queue	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 qc:GPU=1	BIP	0/0/10	0.23	lx-amd64	
gpu1.q@host1 qc:GPU=1	BIP	0/0/10	0.23	lx-amd64	

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 [ "$QUEUE" = "xgpu0.q" ]; then
    echo "Using GPU 0"
fi

if [ "$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
```

queue	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 qc:GPU=0 5 0.55500 gpu.sh	BIP	0/1/10 r	0.05 04/19/2011 08:58:08	lx-amd64	1
gpu1.q@host1 qc:GPU=1	BIP	0/0/10	0.05	lx-amd64	

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
```

3.5 Implementing Preemption Logic

Preemption 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. Preemption 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.

3.5.1 When to Use Preemption

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.

3.5.2 Utilizing Queue Subordination

Queue subordination is defined in the queue configuration, which can be modified through the `qconf -mq <queueenamel>` command. The related attribute for subordination definitions is named `subordinate_list`. The syntax is:

```
<queueenamel>=<slots>, <queueenamel2>=<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:

```
<queue>=<default_slots>, [<queue>@<host>=<slots>]
```

3.5.2.1 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
-----
    4 0.55500 sleep    daniel    r      05/17/2011 15:36:04 low.q@host1
```

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
-----
    4 0.55500 sleep    daniel    S      05/17/2011 15:36:04 low.q@host1
    5 0.55500 sleep    daniel    r      05/17/2011 15:36:14 high.q@host1
```

3.5.3 Advanced Preemption 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.

3.5.3.1 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`:

```
> 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
...
```

Change the scheduler configuration:

```
> qconf -ms
...
queue_sort_method      seqno
...
```

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.

3.6 Integrating Univa Grid Engine With a License Management System

Many applications being run under Univa Grid Engine control are licensed applications, in many cases they come with a license manager application.

Licenses are configured as consumable resources in Univa Grid Engine, see also [Concepts and Components#Expressing Capabilities and Capacities](#) and [Special Activities#Using Consumables](#).

Once a consumable 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 exec host object, for site licenses in the global host, for node locked licenses in the specific exec host.

Univa Grid Engine will keep track of the licenses in use by jobs, no jobs requesting a license will be started if the licenses are exceeded.

The easiest and most precise way of handling licenses is if licenses are only 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 main 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 is combining 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 `exec` host object.

One example of such an external component is the [qlicserver](#).

3.6.1 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.

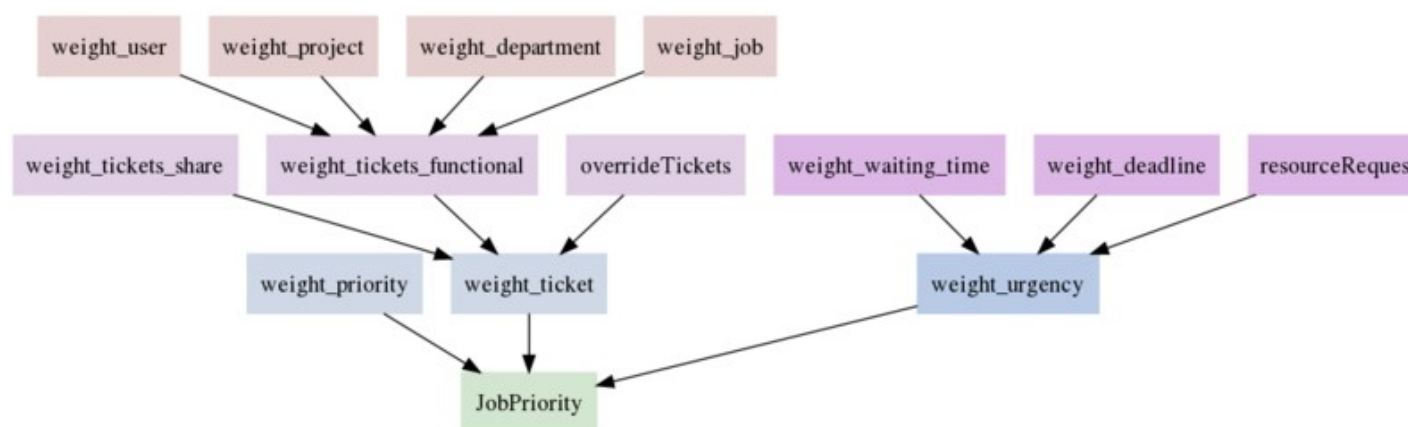
The qlicserver has been developed by Marc Olesen and published under a [Creative Commons License](#).

The following text describes in short the steps required for setting up the qlicserver:

- download the flex-grid tar.gz from <http://olesenm.github.com/flex-grid>
- unpack it to some 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>

3.7 Managing Priorities and Usage Entitlements



The influence of all policy weights on the final priority

Mapping of business rules into the cluster usage is a crucial demand of companies with a computer 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. Within this order, the scheduler dispatches the jobs to the computer nodes, hence jobs at the top of the list have a greater chance of obtaining resources earlier than jobs at the end of the list. Jobs that cannot be dispatched to a computer node due to the lack of resources are deferred to the next scheduling run. This section describes the different policies, which influences 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`.

3.7.1 Fair-Share (Share Tree) Ticket Policy

The fair-share or share-tree policy allows that the usage shares from a Univa Grid Engine cluster can be defined in a hierarchical tree structure (see also `man sharetree`). The main characteristic of this policy is that past job usage is taken into account. Each node of the tree

structure sets a number of tickets, which a certain project or user owns. The share tree is managed with the `qconf` command through various command line arguments:

- `-astree` creates or modifies a share-tree
- `-Astree` creates or modifies share-tree through a given file
- `-clearusage` clears historic usage data
- `-dstnode` deletes a certain node
- `-dstree` deletes the complete share-tree
- `-mstnode` allows to modify a share-tree node
- `-Mstree` creates or modifies a share-tree through a given file
- `-mstree` create or modifies a share-tree
- `-sstnode` shows a specific share-tree node
- `-rsstnode` shows a specific share-tree node and its children
- `-sst` shows a formatted share-tree
- `-smtree` shows the share-tree

A share-tree node consists of an **ID**, the **name**, **type**, **shares** and the **childnodes**. An entry starts with a unique numeric ID. The node can have one of two types: User node (`type=0`) or project node (`type=1`). The name is a string representing the corresponding name of a project or user, while **shares** defines the absolute importance of the node. The childnodes entry is a comma-separated list of the IDs of the children.

In the following example a simple share-tree with 2 users, one with a 70% share and one with a 30% share is created:

```
> qconf -astree
id=0
name=Root
type=0
shares=1
childnodes=1,2
id=1
name=user1
type=0
shares=70
childnodes=NONE
id=2
name=user2
type=0
shares=30
childnodes=NONE
```

The share-tree can be viewed with:

```
> qconf -sst
Root=1
  user1=70
  user2=30
```

In Univa Grid Engine, different policies can be active at the same time. Therefore it is required to give the share-tree policy a **weight** (here: tickets) within these policies. In a default installation there are 0 tickets available. This needs to be modified in the scheduler

configuration (see `man sched_conf`).

```
> qconf -msconf
...
weight_tickets_share          1000
...
```

Now this policy can be tested, but first the previous usage must be cleared, because the policy also takes past usage into account:

```
> qconf -clearusage
```

In order to view the current job list better, the queues can be disabled (in the test cluster). This prevents the dispatch of jobs into the next scheduled run.

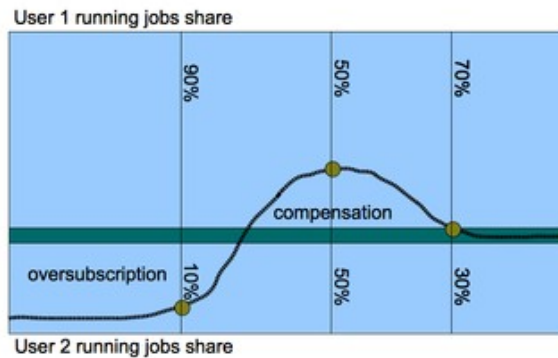
```
> qconf -d all.q
```

If two jobs are submitted, one as **user1** and one as **user2**, the tickets can be seen in a `qstat` output:

```
> qstat -ext -u \*
job-ID  prior   ntckts  name          user      ...  tckts  ovrts  otckt  ftckt  stckt  share queue
-----
    213  0.01000  1.00000  sleep        user1      ...  70     0     0     0     70  0.70
    214  0.00429  0.42857  sleep        user2      ...  30     0     0     0     30  0.30
```

The **stckt** column shows the number of tickets granted through the share-tree policy. In this example the job of **user1** 70 tickets and the job of **user2** 30 tickets are granted. This results in an overall priority (`prior`) of 0.01 versus 0.00429.

When a second job as **user1** is submitted it receives 35 tickets, which is half of the tickets. The third job would receive a third and so on. The second job of **user2** has 15 tickets, therefore all jobs of the first user takes precedence over the jobs of **user2**, within this first round. After a time, accounting comes into play. The nature of the behavior can be seen in the picture **Share Compensation**.



Share Compensation

In this picture two users can be seen, **user 1** with 70 shares and **user 2** with 30 shares. At the beginning are several jobs from **user 1** submitted, but almost none from **user 2**. This means that despite the 70/30 distribution, **user 1** is using 90% of the cluster while **user 2** has just some jobs and therefore a share of 10%. After a period of time, **user 2** begins to submit more jobs. His jobs are now more important (see compensation area) than the **user 1** jobs in order to achieve fairness (the 70/30 distribution) over time.

3.7.1.1 Halftime and Compensation Factor

In the share-tree policy, past usage is taken into account. The influence of historical usage decreases from time to time. The **halftime** attribute in the scheduler configuration (`qconf -msconf`) specifies how long it will take until the past usage loses half of the influence. If the value is 0 the influence will not decrease, other values are time values in hours.

Modification of the halftime value:

```
> qconf -msconf
...
halftime                168
...
```

The `compensation_factor` determines how quickly the past usage is compensated for over time. The values must be between 2 and 10, where 2 is the slowest and 10 the fastest compensation.

Modification of the compensation factor:

```
> qconf -msconf
...
compensation_factor      5.000000
```

...

3.7.2 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 n'th job get 1/n of the calculated number of tickets.

3.7.3 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
-----
  620  1.00000  1.00000  sleep        daniel        mytestproject2 ...   1000    0     0   1000
  615  0.45000  0.45000  sleep        daniel        mytestproject2 ...    450    0     0    450
  616  0.30000  0.30000  sleep        daniel        mytestproject2 ...    300    0     0    300
  617  0.22500  0.22500  sleep        daniel        mytestproject2 ...    225    0     0    225
  618  0.18000  0.18000  sleep        daniel        mytestproject2 ...    180    0     0    180
  619  0.15000  0.15000  sleep        daniel        mytestproject2 ...    150    0     0    150
```

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
-----
  619  0.15000  0.15000  sleep        daniel        mytestproject2 ...   1900   1000   1000   900
  620  1.00000  1.00000  sleep        daniel        mytestproject2 ...    550    0     0    550
  615  0.45000  0.45000  sleep        daniel        mytestproject2 ...    300    0     0    300
  616  0.30000  0.30000  sleep        daniel        mytestproject2 ...    225    0     0    225
  617  0.22500  0.22500  sleep        daniel        mytestproject2 ...    180    0     0    180
  618  0.18000  0.18000  sleep        daniel        mytestproject2 ...    150    0     0    150
```

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.

3.7.4 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.

3.7.4.1 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 behavior 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 means 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).

3.7.4.2 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.

3.7.4.2.1 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
-----
   34 0.60500 1.00000   12215    1000        0   11215 sleep    ... 05/04/2011 14:04:17 0
   35 0.50590 0.00899    1101    1000        0    101 sleep    ... 05/04/2011 14:04:35 0
   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
-----
   34 0.60500 1.00000   24841    1000        0   23841 sleep    ... 05/04/2011 14:04:17
   35 0.50542 0.00425    1101    1000        0    101 sleep    ... 05/04/2011 14:04:35
   33 0.50500 0.00000    1000    1000        0     0 sleep    ... 05/04/2011 14:04:03
```

3.7.4.3 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
```

3.7.5 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.

3.7.5.1 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
```

3.8 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 environments
- 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

3.8.1.1 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.

3.8.1.2 Configuration Parameters of Parallel Environments

Each parallel environment object supports the following set of configuration attributes:

Parallel environment configuration attributes

Attribute	Value specification
<code>pe_name</code>	

	The name of the parallel environment to be used when attaching it to queues or when administering its definition. This name has to be specified by users that explicitly request a certain type of parallelism for jobs.
slots	The number of parallel processes allowed to run in total under the parallel environment concurrently.
user_lists	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 <code>xuser_lists</code> parameter described below.
xuser_lists	The <code>xuser_lists</code> 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 <code>xuser_lists</code> parameter is set to <code>NONE</code> (the default) any user has access. If a user is contained both in an access list enlisted in <code>xuser_lists</code> and <code>user_lists</code> the user is denied access to the parallel environment.
start_proc_args	<p>This parameter defines the command line of a start-up procedure for the parallel environment. The keyword <code>NONE</code> 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 to set up the parallel environment corresponding to its needs. The syntax for the parameter value is:</p> <pre>[username@]path [arg ...]</pre> <p>The optional <code>username</code> prefix specifies the user under which this procedure is started. The standard output of the start-up procedure is redirected to the file <code>NAME.poJID</code> in the job's working directory, with <code>NAME</code> being the name of the job and <code>JID</code> which is the job's identification number. Likewise, the standard error output is redirected to <code>NAME.peJID</code>. 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:</p> <p>\$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 hostname, 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.</p> <p>\$host The name of the host on which the startup or stop</p>

	<p>procedures are started.</p> <p><code>\$job_owner</code> The user name of the job owner.</p> <p><code>\$job_id</code> Univa Grid Engine's unique job identification number.</p> <p><code>\$job_name</code> The name of the job.</p> <p><code>\$pe</code> The name of the parallel environment in use.</p> <p><code>\$pe_slots</code> Number of slots granted for the job.</p> <p><code>\$processors</code> The processor's string as contained in the queue configuration of the primary queue were the parallel job is started (master queue).</p> <p><code>\$queue</code> The cluster queue of the queue instance were the parallel job is started.</p>
<code>stop_proc_args</code>	<p>The invocation command line of a shutdown procedure for the parallel environment. Similar to <code>start_proc_args</code> this method is executed on the execution host. The keyword <code>NONE</code> 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 <code>start_proc_args</code>.</p>
<code>allocation_rule</code>	<p>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:</p> <p><code>\$fill_up</code> 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.</p> <p><code>\$round_robin</code> 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.</p>

	<p>Positive number or <code>\$pe_slots</code></p> <p>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 <code>\$pe_slots</code> is used, the full range of processes as specified with the <code>qsub -pe</code> 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.</p>
<code>control_slaves</code>	<p>This parameter can be set to <code>TRUE</code> or <code>FALSE</code>. 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. <code>FALSE</code> is the default for this parameter.</p>
<code>job_is_first_task</code>	<p>The <code>job_is_first_task</code> parameter can be set to <code>TRUE</code> or <code>FALSE</code>. A value of <code>TRUE</code> indicates that the Univa Grid Engine job script will also start one of the tasks of the parallel application, while a value of <code>FALSE</code> indicates that the job script and its child processes are not part of the parallel program. In this case the the number of slots reserved for the job is the number of slots requested with the <code>-pe</code> switch of the submit application plus one additional slot).</p>
<code>urgency_slots</code>	<p>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:</p> <p>Positive number The specified number is directly used as prospective slot amount.</p> <p><code>min</code> The slot range minimum is used as the prospective slot amount. If no lower bound is specified, range 1 is assumed.</p> <p><code>max</code> 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.</p> <p><code>avg</code> The average of all numbers occurring within the job's parallel range request is assumed.</p>
<code>accounting_summary</code>	

	This parameter is only checked if <code>control_slaves</code> is set to <code>TRUE</code> . In this case accounting information is available for every single slave task of a parallel job. These parameters can be set to <code>TRUE</code> 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.

3.8.1.3 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 hostname 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.

3.8.1.4 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 where 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 undelaying 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.

Submit parameters influencing parallel jobs

Attribute	Value specification
<code>-pe</code>	<code>qsub</code> , <code>qsh</code> , <code>qrsh</code> , <code>qlogin</code> or <code>qalter</code> switch is followed by a parallel environment specification of the following format: <code>pe_name pe_min[-[pe_max]], [-]pe_max</code> where <code>pe_name</code> specifies the parallel environment to instantiate and <code>pe_min</code> and <code>pe_max</code> specify the minimum or maximum number of slots that might be used by the parallel application.
<code>-masterq</code>	This parameter is available for <code>qsub</code> , <code>qsh</code> , <code>qrsh</code> and <code>qalter</code> 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.
<code>-v</code> and <code>-V</code>	These parameters are available for <code>qsub</code> , <code>qsh</code> , <code>qrsh</code> , <code>qlogin</code> and <code>qalter</code> . 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.

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 `big.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.

3.8.3.1 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`
Adda 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.

3.8.3.2 Configuration Parameters for Checkpointing Environments

Each checkpointing environment supports the following set of configuration attributes:

Checkpointing environment configuration attributes

Attribute	Value specification
<code>ckpt_name</code>	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.
<code>interface</code>	<p>The type of checkpointing to be used. Currently, the following values are supported:</p> <p><code>hibernator</code> The Hibernator kernel level checkpointing is interfaced.</p> <p><code>cpr</code> The SGI kernel level checkpointing is used.</p> <p><code>cray-ckpt</code> The Cray kernel level checkpointing is assumed.</p> <p><code>transparent</code> Univa Grid Engine assumes that jobs that are submitted with reference to this checkpointing environment use a public domain checkpointing environment such as Condor.</p> <p><code>userdefined</code> Jobs submitted with reference to this type of checkpointing interface perform their private checkpointing method.</p>

	<p><code>application-level</code></p> <p>In this case all interface commands specified with this object will be used. One exception is the <code>restart_command</code>. Instead of that command the job script itself is restarted.</p>
<code>ckpt_command</code>	A command line type command string to be executed by Univa Grid Engine in order to initiate a checkpoint.
<code>migr_command</code>	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.
<code>restart_command</code>	A command line type command string to be executed by Univa Grid Engine when restarting a previously checkpointed job.
<code>clean_command</code>	A command line type command string to be executed by Univa Grid Engine in order to cleanup after a checkpointing application has finished.
<code>ckpt_dir</code>	A file system location to which checkpoints of potentially considerable size should be stored.
<code>ckpt_signal</code>	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 <code></code>kill -l</code> command or an integer number which must be a valid signal on the system used for checkpointing.</code>
<code>when</code>	<p>Defines the points in time when checkpoints are expected to be generated. Valid values for this parameter are composed by the letters <code>s</code>, <code>m</code>, <code>x</code> and <code>r</code> and any combination thereof without any separating character in between. The same letters are allowed for the <code>qsub -c</code> command which will overwrite the definitions in the checkpointing environment used. The meaning of the letters is defined as follows:</p> <p><code>s</code> A job is checkpointed, aborted and if possible, migrated if the corresponding <code>sge_execd</code> is shut down where the job is executed.</p> <p><code>m</code> Checkpoints are generated periodically at the <code>min_cpu_interval</code> defined by the queue in which a job is running.</p> <p><code>x</code> A job is checkpointed, aborted and if possible migrated as soon as the job gets suspended (manually as well as automatically).</p> <p><code>r</code> A job will be rescheduled (not checkpointed) when the host on which the job currently runs goes into an unknown state.</p>

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.

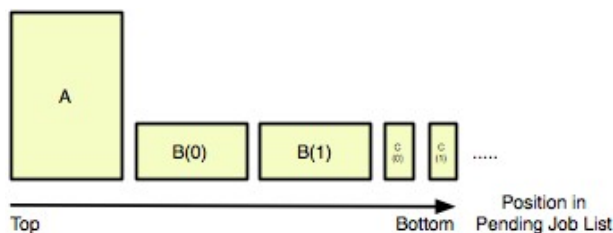
A resource reservation is a job-specific reservation created by the scheduler component for a pending job. During the reservation the resources for jobs of lower priority are blocked so that "job starvation" does not occur.

Advanced 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 advanced reservation causes the requested resources to be blocked for other jobs that are submitted later on.

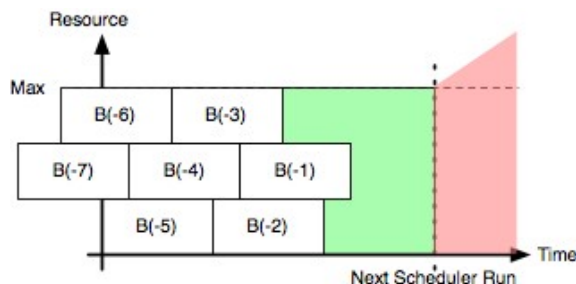
Backfilling is the process of starting jobs of the priority list despite pending jobs of higher priority that might own a future reservation with the same requested resources. Backfilling has only a meaning in the context of resource reservation and advanced reservation.

3.8.4.1 Reservation and Backfilling

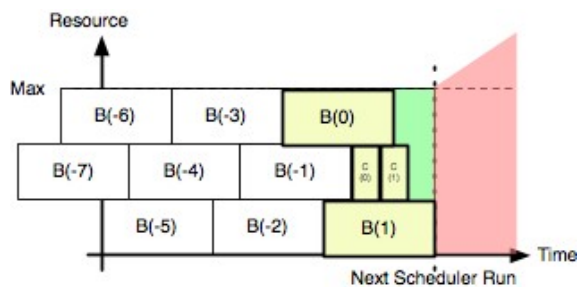
Resource reservation can be used to guarantee resources dedicated to jobs in job priority order. A good example to explain the problem solved with resource reservation and backfilling is the "large parallel job starvation problem". In this scenario there is one pending job of high priority (possibly 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.



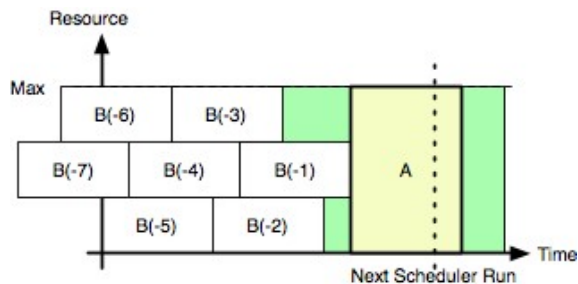
The cluster where the jobs are waiting to be scheduled is already full with running B(i) jobs.



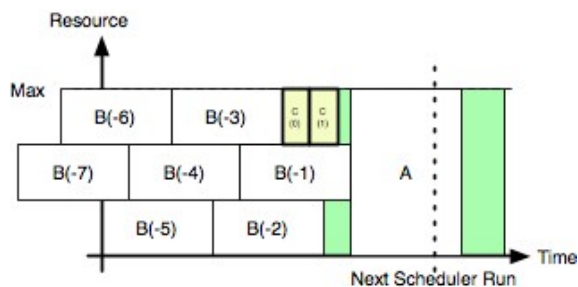
Without resource reservation an assignment for A cannot be guaranteed assume the steam of B(i) and C(i) jobs does not stop - even if job A actually has higher priority than the B(i) and C(i) jobs. Without reservation the scheduler sees only the green area in the resource diagram that is too small for 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:

Commandline parameter that influence reservation

Parameter	Description
-R	This submit option is available for <code>qsub</code> , <code>qrsh</code> , <code>qsh</code> , <code>qlogin</code> and <code>qalter</code> . 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 <code>B(i)</code> job reservations for the sake of guaranteeing the job A resource assignment. The only job that needs to be submitted with the <code>-R y</code> is job A. This means all <code>B(i)</code> jobs can be submitted with the <code>-R n</code> . without actually endangering the reservation for A. Default is <code>-R n</code> if none other is specified.
<code>-now</code>	Although it might be requested, reservation is never done for immediate jobs using <code>-now yes</code> option.

Parameters in scheduler configuration that influence reservation

Parameter	Description
<code>max_reservations</code>	For limiting the absolute number of reservations made during a scheduling interval, the <code>max_reservations</code> parameter in the scheduler configuration can be used by Univa Grid Engine administrators. E.g. when <code>max_reservations</code> 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.
<code>MONITOR</code>	If <code>MONITOR</code> is added to the <code>params</code> 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 <code>\$SGE_ROOT/\$SGE_CELL/common/schedule</code> .
<code>DURATION_OFFSET</code>	If <code>DURATION_OFFSET</code> 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 <code>-l h_rt=...</code> or <code>-l s_rt=...</code> or <code>default_duration</code> 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 <code>schedule_interval</code> , the time it takes to deliver a job to <code>sge_execd</code> , the time <code>prolog</code> and <code>starter_method</code> in queue configuration need and the <code>start_proc_args</code> in parallel environments may be affected. The delays after a job's actual run include delays due to a forced job termination (<code>notify</code> , <code>terminate_method</code> or various checkpointing methods), procedure runs after actual job completion such as <code>stop_proc_args</code> in parallel configurations or <code>epilog</code> in queues and the delay until a new <code>schedule_interval</code> . If the offset is too low, resource reservations can be delayed repeatedly.

3.8.4.2 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.

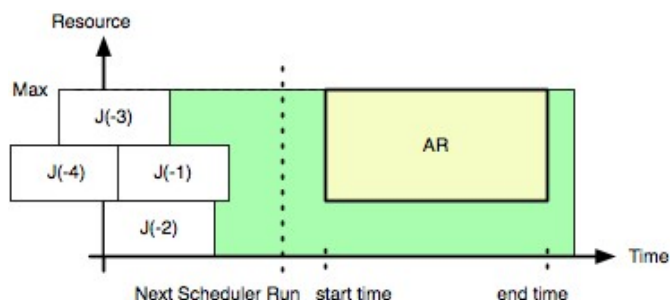
An advance reservation is defined as a reservation of resources done by the scheduler due to a user or administrator request. This reservation is made at the beginning independent

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:

States of an advance reservation

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 where reserved are not available in the cluster anymore.

The following commands address advance reservations or influence them:

- `qsub`
Used to submit a new advance reservation. Returns the identifier `{ar_id}` that is needed as parameter for other commands
- `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 `qrs`, 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.

qrs command line switches

State	Description
<code>-a {date_time}</code>	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 <code>-a {date_time}</code> is: <code>[[CC]YY]MMDDhhmm[.ss]</code> where <code>CC</code> denotes the century, <code>YY</code> the year, <code>MM</code> the month, <code>DD</code> the day, <code>hh</code> the hour, <code>mm</code> the minutes and <code>ss</code> the seconds when the job could be started.
<code>-A {account_string}</code>	Identifies the account to which the resource reservation of the AR should be charged.
<code>-ckpt {ckpt_name}</code>	Selects the checkpointing environment the AR jobs may request. Using this option guarantees that only queues providing this checkpointing environment will be reserved.
<code>-d {time}</code>	Defines the duration of the advance reservation. The use of this switch is optional if an end date and time is specified with <code>-e</code>
<code>-e {date_time}</code>	Defines the end date and time of the advance reservation. The use of this switch is optional if <code>-d</code> is requested. Format of <code>{date_time}</code> is the same as for <code>-a</code> .
<code>-he {y_or_n}</code>	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 <code><code>-he y</code>) then the AR stays usable with the remaining resources. By default, soft error handling is used.
<code>-l {requests}</code>	The created AR will provide the given resources specified in <code>{requests}</code> . Format of <code>{requests}</code> is the same as for <code>qsub -l</code> .
<code>-m {occasion}</code>	Defines under which circumstances mail is sent to the AR owner or to the users defined myth <code>-M</code> option. <code>{occasion}</code> can be a letter combination of the letters <code>b</code> , <code>e</code> and <code>a</code> or the letter <code>n</code> where <code>b</code> is a beginning mail, <code>e</code> an end mail, <code>a</code> a mail when the AR goes into error state and <code>n</code> will disable sending of any mail. By default, no mail will be sent.
<code>-now {y_or_n}</code>	This option impacts the queue selection for reservation. With <code>y</code> , only queues with the type <code>INTERACTIVE</code> assigned will be considered for reservation and <code>n</code> is default.
<code>-N {name}</code>	The name of the advance reservation.

<code>-pe {pe_name} {pe_range}</code>	Selects the parallel environment the AR jobs may request, Using this option guarantees the queues providing this parallel environment will be reserved.
<code>-w {level}</code>	Specifies the validation level applied to the AR request. <code>v</code> does not submit the AR but prints an validation report whereas <code>e</code> means that an AR should be rejected if requirements cannot be fulfilled. <code>e</code> is the default.

3.8.5 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:

1. the default request file located in `$SGE_ROOT/$SGE_CELL/common`
2. the user default request file `.sge_request` located in `$HOME` of the submit user
3. 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 submit 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
```

3.8.6 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.

3.8.6.1 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/Server JSV 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 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.

3.8.6.1.1 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:

1. `-jsv` used as command line parameter with the submit client
2. `-jsv` used in the `.sge_request` file located in the current working directory where the submit client is executed
3. `-jsv` used in the `.sge_request` file located in `$HOME` of the user that tries to submit a job
4. `-jsv` used in the global `sge_request` file in `$SGE_ROOT/$SGE_CELL/default/common`
5. `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.

3.8.6.1.2 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 informations 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/`sges_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.

3.8.6.1.3 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.

3.8.6.1.3.1 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_start()  
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 remainder to see if the specified slots value was a multiple of 16.
- The job is rejected in line 6.

3.8.6.1.3.2 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:

- `jsh_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 `jsh_is_param(param_name)` can be used to check if the parameter list is not available.
- `jsh_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. `jsh_is_sub_param(param_name)` can be used to check this.
- `jsh_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 `jsh_sub_is_param(param_name, variable_name)`
- `jsh_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=`jsh_get_param l_hard`
02 if [ "$l_hard" != "" ]; then
03     has_soft_lic=`jsh_sub_is_param l_hard soft_lic`
04
05     if [ "$has_soft_lic" = "true" ]; then
06         jsh_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).

3.8.6.1.3.3 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.

3.8.6.1.3.4 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 send 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 send 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 send 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.

3.8.6.1.3.5 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 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 submit the job. Command line clients like `qsub` will print this message to notify the user that the submission has failed.

3.8.6.1.3.6 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.

3.8.6.1.4 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.

JSV Job Parameter Names

Parameter	Description
a	If a job has a specific start time and date at which it is eligible for execution (specified with <code>qsub -a</code> at the commandline) then the corresponding value is available in JSV as parameter with the name <code>a</code> . The value of this parameter has the following format: <code>[[CC]YY]MMDDhhmm[.ss]</code> where <code>CC</code> denotes the century, <code>YY</code> the year, <code>MM</code> the month, <code>DD</code> the day, <code>hh</code> the hour, <code>mm</code> the minutes and <code>ss</code> the seconds when the job could be started.
ac	The value for the <code>ac</code> parameter represents the job context of a job as it is specified at the command line with the command line switches <code>-ac</code> , <code>-dc</code> and <code>-sc</code> . The outcome of the evaluation of all three switches will be

	passed to JSV as the list parameter named <code>ac</code> . It is possible within JSV scripts to modify this list with the <code>jsv_sub_*_param()</code> functions.
<code>ar</code>	The <code>ar</code> 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 <code>-ar</code> switch. The value of <code>ar</code> can be changed in JSV as long as the new value is a valid advance reservation id.
<code>b</code>	If the parameter named <code>b</code> is available in JSV this shows that a job was submitted as binary job e.g with <code>-b</code> switch at the command line. The value in this case is <code>yes</code> . 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.
<code>c_interval</code> <code>c_occasion</code>	The command line switch <code>-c</code> of <code>qsub</code> 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 <code>c_interval</code> and when certain occasion are specified through characters then this letter is available through the parameter <code>c_occasion</code> . It is possible to change both values in JSV. Note that a change of <code>c_occasion</code> will automatically override the current value of <code>c_interval</code> and vice versa. Valid values for <code>c_occasion</code> are the letters <code>n</code> , <code>s</code> , <code>m</code> and <code>x</code> where <code>n</code> disables checkpointing, <code>s</code> triggers a checkpoint when an execution daemon is shut down, <code>m</code> checkpoint at minimum CPU interval and <code>x</code> checkpoints when the job gets suspended. The time value for <code>c_occasion</code> has to be specified in the format <code>hh:mm:ss</code> where <code>hh</code> denotes hours, <code>mm</code> denotes minutes and <code>ss</code> the seconds of a time interval between two checkpointing events.
<code>ckpt</code>	The <code>ckpt</code> parameter is set for checkpointing jobs and contains the name of the checkpointing environment that can be defined at commandline with the <code>-ckpt</code> switch.
<code>cwd</code>	The value of the <code>cwd</code> 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 <code>-cwd</code> switch.
<code>display</code>	The value of <code>display</code> is used by <code>xterm</code> to contact the X server. At the command line the value for this parameters can be set in <code>qsh</code> and <code>qrsh</code> with the <code>-display</code> switch. The format of the display value has always to start with a hostname (e.g <code>hostname:1</code>). Local display names (e.g. <code>:13</code>) cannot be used in grid environments. Values set with the <code>display</code> variable in JSV will overwrite settings from the submission environment and environment variable values specified with <code>-v</code> command line option.
<code>dl</code>	The <code>dl</code> 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 <code>-dl</code> switch. If available the value will have the same format as the <code>a</code> parameters that specifies the start time of a job.
<code>e</code>	The <code>e</code> 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 <code>-e</code> switch.
<code>h</code>	The value of the <code>h</code> parameter indicates that a job was submitted in user hold state (e.g. with <code>qsub -h</code>). In this case the parameter is available and it will be set to <code>u</code> . To change this the parameter can be set to <code>n</code> .
<code>hold_jid</code>	<p>The <code>hold_jid</code> parameter contains job dependency information of a job. It is available when a job was submitted with <code>-hold_jid</code> command line switch.</p> <p>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.</p>
<code>hold_jid_ad</code>	The <code>hold_jid_ad</code> parameter defines or redefines array job interdependencies. It is available when a job was submitted with <code>-hold_jid_ad</code> 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.
<code>i</code>	The <code>i</code> 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 <code>-i</code> switch.
<code>j</code>	Similar to the <code>-j</code> command line switch the <code>j</code> 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 <code>y</code> . To change this, the value can get set to <code>n</code> .
<code>js</code>	Defines or redefines the job share of a job relative to other jobs. If the corresponding <code>-js</code> 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.
<code>l_hard</code> <code>l_soft</code>	At the command line job resource requests are specified with the <code>-l</code> switch. This switch can be used multiple times also in combination with the switches <code>-hard</code> and <code>-soft</code> 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 <code>l_hard</code> and <code>l_soft</code> . 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 <code>jsv_sub*_param()</code> functions.
<code>m</code>	The value of the <code>m</code> parameter defines or redefines when Univa Grid Engine sends mail to the job owner. Format is similar to the command line switch <code>-m</code> of the <code>qsub</code> command. <code>n</code> means that there is no mail sent and different letter combinations of the letters <code>b</code> , <code>e</code> , <code>a</code> and <code>s</code> can be used to define when mail is sent where <code>b</code> means that mail is sent at the beginning of a job, <code>e</code> at the end of a job and <code>a</code> when the job is aborted or rescheduled.
<code>M</code>	<code>M</code> 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 <code>jsv_sub*_param()</code> 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 <code>-masterq</code> commandline switch. In JSV the list can be accessed with the <code>jsv_sub_*_param()</code> script functions.
notify	Jobs where the <code>notify</code> parameter is available in JSV and where it is set to <code>y</code> will receive a notify signal immediately before a suspend or kill signal will be delivered. If <code>-notify</code> was not used during submission of a job then the <code>notify</code> parameter will not be available.
now	Not available in JSV.
N	The value of <code>N</code> is the job name of the job to be submitted.
o	The <code>o</code> 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 <code>-o</code> switch.
p	The <code>p</code> 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 <code>qsub</code> , the <code>-pe</code> 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 <code>pe_name</code> , <code>pe_min</code> and <code>pe_max</code> show parts of that specification. <code>pe_name</code> is the name of the parallel environment. <code>pe_min</code> and <code>pe_max</code> specify the biggest and smallest slot number used in the slot specification. Due to an error in the JSV module of Univa Grid Engine it is only possible to change the <code>pe_name</code> with JSV in the moment.
pty	<p>This parameter is only available in Univa Grid Engine 8.0.1 and above (see <code>UNIVA_EXTENSIONS</code> pseudo parameter below).</p> <p>The <code>-pty</code> switch of <code>qcrsh</code> and <code>qsh</code> 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 <code>pty</code>. If the command line switch is omitted then then this parameters has the value <code>u</code> which means unset. Client application and executed job will use the default behavior. <code>y</code> means that the use of a pseudo terminal is enforced and <code>n</code> 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>
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 <code>-P</code> command line parameter.
q_hard q_soft	The <code>-q</code> switch at the command application can be combined with the <code>-hard</code> and <code>-soft</code> 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 <code>q_hard</code> and <code>q_soft</code> . Both of them can be changed using the <code>jsv_sub_*_param()</code> script functions.

R	If the <code>R</code> parameter is available and set to <code>y</code> 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 <code>n</code> .
r	<code>r</code> parameter is available and set to <code>y</code> . To overwrite this the value can be changed to <code>n</code> .
shell	The parameter <code>shell</code> is defined and set to <code>y</code> if a command shell should be used to start the job. To disable this JSV value has to be changed to <code>n</code> .
sync	<p>This parameter is only available in Univa Grid Engine 8.0.1 and above (see <code>UNIVA_EXTENSIONS</code> pseudo parameter below).</p> <p>When a command line application is used with the <code>-sync</code> command-line switch then within client and server JSV the parameters with the name <code>sync</code> will be available and it will be set to <code>y</code>. The <code>sync</code> 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.</p>
S	The <code>S</code> parameters specifies the interpreting shell for the job.
t_min t_max t_setp	The <code>-t</code> parameter of <code>qsub</code> submits an array job. The task range specification is available in JSV via three parameters: <code>t_min</code> , <code>t_max</code> and <code>t_step</code> . All three values can be changed.
terse	This parameter is only available in Univa Grid Engine 8.0.1 and above (see <code>UNIVA_EXTENSIONS</code> pseudo parameter below). When a command line application is used with the <code>terse</code> -switch then the parameter named <code>terse</code> will be available in client and server JSV scripts and it will be set to <code>y</code> . If this parameters is set to <code>n</code> 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.
⚡	There is no <code>v</code> 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 <code>jsv_send_env()</code> in <code>jsv_on_start()</code> . After that the <code>jsv_*_env()</code> functions can be used to access the job environment.
V	<p>This parameter is only available in Univa Grid Engine 8.0.1 and above (see <code>UNIVA_EXTENSIONS</code> pseudo parameter below).</p> <p>The <code>v</code> parameter will be available in client and server JSV scripts and it will have the value <code>y</code> when the <code>-v</code> command line switch was used during the submission of a job. This indicates that the full set of environment variables that where set in the submission environment can be accessed from JSV. If this parameter is not available or when it is set to <code>n</code> 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 <code>-v</code> command line parameter.</p>
wd	See <code>cwd</code>

Additionally to the job parameters JSV provides a set of pseudo parameters

JSV Pseudo Parameter Names

Parameter	Description
CLIENT	The value of the <code>CLIENT</code> is either <code>qmaster</code> 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 <code>qsub</code> , <code>qrsh</code> , <code>qsh</code> , <code>qlogin</code> or <code>qmon</code> . In case of DRMAA clients the string <code>drmaa</code> is used. This value is read-only. It cannot be changed by JSV.
CMDARG{ <i>i</i> }	Command line arguments of the job script will be available within JSV via multiple <code>CMDARG{<i>i</i>}</code> parameters where <code>{<i>i</i>}</code> is replaced with by the number of the position where the argument should appear. <code>{<i>i</i>}</code> is a number in the range starting from 0 to <code>CMDARGS - 1</code> . This means that the first argument will be available through the parameter <code>CMDARG0</code> .
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 <code>CMDNAME</code> contains the command name of the binary to be executed, for non-binary jobs the full path to the job script is specified.
CONTEXT	The <code>CONTEXT</code> might have two values. <code>client</code> or <code>master</code> depending on which client host the JSV is currently executed. It is not possible to change this value.
GROUP	The value of <code>GROUP</code> 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 <code>UNIVA_EXTENSIONS</code> pseudo parameter below). Within server JSV's the read-only parameter <code>SUBMIT_HOST</code> is available. This parameter contains the hostname where the submit application is executed.
JOB_ID	This variable is not available when <code>CONTEXT</code> is <code>client</code> (client JSV). In case of server JSV the value of <code>JOB_ID</code> 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 <code>UNIVA_EXTENSIONS</code> 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 <code>n</code> then these extensions to JSV are missing (Open Source version of Grid Engine). In this case it is not possible to access following parameters: <code>pty</code> , <code>sync</code> , <code>terse</code> , <code>V</code> and <code>SUBMIT_HOST</code> .
USER	

	The value of <code>USER</code> is the Unix user name of the user who submitted the job. Cannot be changed by JSV.
<code>VERSION</code>	Shows the <code>VERSION</code> of the implementation of the JSV communication protocol. <code>VERSION</code> is always available in JSV and it is not possible to change the value. The format of the value is <code>{major}.{minor}</code> . Since the first implementation of JSV the communication protocol has not been changed so that the current <code>VERSION</code> is still <code>1.0</code>

3.8.6.2 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.

3.8.7 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`.

3.8.7.1 Example: Enabling Core Binding on Host `host1`

```
> qconf -mconf host1
mailer                               /bin/mail
...
execd_parameter                      ENABLE_BINDING=true
```

3.8.8 The Univa Grid Engine Hadoop Integration

Univa Grid Engine provides an integration with Apache Hadoop. In a nutshell, it enhances Hadoop with enterprise level accounting and reporting features, provides expanded policy control for Hadoop and integrates Hadoop into shared resource pools which are used for Hadoop calculations as well as other workloads.

3.8.8.1 Brief Introduction to Apache Hadoop

Apache Hadoop is an innovative and popular framework for developing and running large and scalable data analytics applications such as data mining or parameter studies. It is an open implementation of the map-reduce paradigm and comes with HDFS, the Hadoop Distributed File System, for high throughput access to distributed data.

There are some well-known problems to Hadoop, however. If Hadoop is being run on Hadoop-dedicated resources for Hadoop-based applications, there are few problems, if any. But if the intentions are to share such a resource pool across Hadoop applications as well as other workloads, then Hadoop behaves badly. It will not recognize that other workloads are attempting to utilize the same resources and will assume that all resources are under Hadoop control. This leads to overloaded nodes and a slowdown of applications caused by congestion.

One can choose to set aside resources specifically for Hadoop at the expense of using resources optimally; ultimately, is harder to integrate Hadoop applications with other workloads.

However, even a resource pool used for nothing but Hadoop applications will encounter a number of issues. Hadoop has limited scheduling features. It supports prioritization and a fair-share policy but it does not, for instance, allow the control of resource consumption per user, per group or per project. Hadoop also does not offer true accounting, i.e. it does not track resource usage over periods of time and it does not track that usage per individual jobs or the users, user groups and projects. It does not conduct aggregated accounting if the resource pool is shared across Hadoop and non-Hadoop applications.

3.8.8.2 Benefits of Using Hadoop with Univa Grid Engine

Using Hadoop together with Univa Grid Engine is beneficial if the intention is to utilize the resource pool not only for Hadoop calculations but also for sharing those resource with other workloads or to integrate Hadoop applications with non-Hadoop applications. Even if only using Hadoop-based applications there is still a benefit in using Univa Grid Engine together with Hadoop via the advanced policy management and accounting features of Univa Grid Engine. Hadoop also operates while embedded in a private, hybrid or public cloud framework

created by Univa Grid Engine and [UniCloud](#).

Univa Grid Engine treats Hadoop applications as a 'tightly integrated' parallel job. In doing so, Grid Engine has full control over all aspects of the Hadoop framework. Hadoop thereby inherits all the benefits which Univa Grid Engine, as an advanced enterprise-grade resource management system, provides such as versatile policy control and full accounting and reporting. It remains in friendly coexistence with other workloads in the Univa Grid Engine cluster, including other parallel applications, e.g. using MPI, or throughput applications or even interactive work under Univa Grid Engine control.

3.8.8.3 How Hadoop is Integrated with Univa Grid Engine

- Univa Grid Engine keeps track of the layout of HDFS data blocks and HDF data racks as they are distributed across the nodes in the Hadoop cluster. This is accomplished by a load sensor. This load sensor will reflect the data layout in HDFS in the `hdfs_primary_rack`, `hdfs_secondary_rack`, and in one or more `hdfs_blk*` complexes. See [here](#) for more information on load sensors and complexes.
- Users submit job requests with access to the data needed in the form of an HDFS absolute path. A [Job Submission Verifier](#) (JSV) has been created which will translate this HDFS absolute path into requests to `hdfs_primary_rack`, `hdfs_secondary_rack`, and `hdfs_blk*` complexes for each job. This ensures that only the nodes selected for the job with resident HDFS data blocks are required.
- A Hadoop Parallel Environment (PE) is available in tight integration with Hadoop, including the queues in which Hadoop jobs are processed. Users submit Hadoop jobs to that PE. See [here](#) for more information regarding PEs.
- Once appropriate nodes are available, Hadoop jobs are started in a MapReduce cluster which is set up on those nodes for the job. This will create a Hadoop configuration directory under `$TMPDIR/conf` with properly modified `mapred.xml` as well as `slaves` and `master` files.
- The Hadoop PE is configured to start a Hadoop *job tracker* on the master node of the parallel Hadoop job and a *task tracker* on each of the slave nodes. It will, furthermore, put the URI and the URL of the *job tracker* into the *context* of the parallel Hadoop job. See [Using the Job Context](#) for more information on job contexts.
- Univa Grid Engine will have full control over the processes belonging to the Hadoop jobs and its tasks. It will collect all accounting data and will be able to complete all processes in case a job is aborted via a `qdel` command.

Note, however, that the `suspend` and `resume` commands are currently not supported by Hadoop. `Susp`

3.8.8.4 Installing the Hadoop Integration

3.8.8.4.1 Installation Prerequisites

The following things should be taken into account before the installation of the Hadoop integration:

- Hadoop is installed on all nodes of a Univa Grid Engine cluster. If installing it only on part of the nodes, then the Hadoop parallel environment is restricted to the respective nodes only. By default, the Hadoop parallel environment is created to encompass all nodes of a cluster which must be modified after installation. See [here](#) for how to manipulate parallel environments.
- An appropriate Java run-time environment must be available (Java SE 6 or higher). If this not be possible, as in the item above, the Hadoop parallel environment should be restricted to encompass only those nodes which are compliant to this requirement.
- The load sensor monitoring the HDF data layout must be run with HDFS super user privileges. Load sensors must be run in Univa Grid Engine either under the `root` account or as *admin user*. HDFS needs to be installed and configured to allow either `root` or the *admin user* to have HDFS super user access privileges. See the [HDFS Permissions Guide](#) for more information.

3.8.8.4.2 Installing the Hadoop Integration Package

1. The integration package is located in the top level Univa Grid Engine installation directory and is called `hadoop`. A copy of this directory should be created and the rest of the installation steps executed there so follow-up updates of the Univa Grid Engine software do not automatically overwrite files important for the Hadoop integration configuration. The copy of the `hadoop` directory needs to be accessible by all nodes of the Hadoop cluster being set up for use together with Univa Grid Engine. This is most easily accomplished if the creation of this copy on a shared file system has with access for all nodes. The following steps assume this is the case.
2. Make sure the files in the `conf` subdirectory are correct and permit access to HDFS.
3. The following files are created for each Hadoop job and thus must be removed at this point: `mapred.xml`, `slaves` and `masters`.
4. Now install the Hadoop integration by running `./setup.pl -i`. This will carry out the following steps:
 1. Creation of the `hdfs_primary_rack`, `hdfs_secondary_rack`, and `hdfs_blk*` complexes.
 2. Creation of the parallel environment called `hadoop`.
 3. Adding the load sensor monitoring the HDFS data layout to the *global* host.
5. The `hadoop` parallel environment now needs to be configured to allow Hadoop jobs to be started on cluster nodes. This is accomplished by adding queues to the parallel environment configuration of the `hadoop` PE. See [here](#) for more information on how to setup PEs.

If a network file sharing program is not used, then one copy of the `hadoop` directory should be created. Set it up as described above and copy it to all nodes in the Hadoop cluster. However, the installation script `setup.pl` only needs to be executed once on an administration host.

The copy or copies of the `hadoop` directory as `<hadoop_site>` are described below.

3.8.8.4.3 Verifying the Hadoop Integration

1. Run `qghost -F hdfs_primary_rack` to check whether the HDFS data layout has been reported correctly by the load sensor. Due to load reporting intervals and number of nodes, several minutes may pass before the `hdfs_primary_rack` complex is populated for all nodes. If, after waiting for several minutes, there are still nodes missing to report values for `hdfs_primary_rack` then check the installation of that host as described under [Hadoop Integration Troubleshooting](#).
2. Run a job which translates the `hdfs_input` absolute HDFS file path into racks and data blocks to check whether it runs correctly. The translation will be executed by the JSV script located in the Hadoop configuration directory `<hadoop_site>`. A suitable job to tests this would be

```
qsub -h -jsv <hadoop_site>/jsv.sh -l hdfs_input=<absolute HDFS path> $SGE_ROOT/examples/jobs/sleeper.sh
```

Afterwards, check for whether the `hdfs_input` request has been replaced with requests for `hdfs_primary_rack`, `hdfs_secondary_rack`, and several `hdfs_blk*` complexes. Do this by running the `qstat -j <job_id>` command and check the output. The job may be deleted when that is done.

3. Now test whether the `hadoop` parallel environment is working properly and, in particular, whether a job tracker for the job has been started and published in its URI and URL. To do this, submit a very simple but reasonably long-running job to the `hadoop` PE, for example:

```
qsub -pe hadoop 1 $SGE_ROOT/examples/jobs/sleeper.sh 400
```

Now use `qstat -j <job_id>` again and search the *context* section of the output for the entries

- ◆ `hdfs_jobtracker`: The URI of the MapReduce job tracker used for manipulating this job tracker together with the `mapred.xml` file.
- ◆ `hdfs_jobtracker_admin`: The URL of the administrative web interface of the job tracker. The web browser can be pointed to the URL to display the administrative web interface for this particular job. In our example, the web page shows that the number of task trackers for this job is 1.
That job can be deleted with `qdel <job_id>` again and repeated on the same test with larger numbers of tasks, e.g. using a `-pe hadoop 3` request in the above job submission example and check whether the number of tasks tracker is indeed 3.

4. A more serious Hadoop test example for which a little shell script `hadoop.sh` may be created is as follows: `#!/bin/sh`

```
$HADOOP_HOME/bin/hadoop --config $TMPDIR/conf fs -lsr  
<data_path_in_HDFS>
```

With `<data_path_in_HDFS>` being a path in HDFS which exists. Next, submit this script to Univa Grid Engine via the command:

```
qsub -pe hadoop 1 hadoop.sh
```

Wait for completion of the job and check the output file. It should contain the complete listing of the directory `<data_path_in_HDFS>`.

5. Finally a parallel Hadoop example can be run. Again, create a corresponding job script first, this time called `hadoop-grep.sh`:

```
#!/bin/sh  
$HADOOP_HOME/bin/hadoop --config $TMPDIR/conf jar  
$HADOOP_HOME/hadoop-*/examples.jar grep <input_path>
```

<output_path> <grep_pattern>

Check the [Hadoop Quickstart Guide](#) for information on how to use the *grep* example and for how to choose values for `input_path`, `<output_path>` and `<grep_pattern>`.

The job tracker's administrative web interface can be used to monitor the job and the output generated by the job.

3.8.8.4.4 Troubleshooting the Hadoop Integration

3.8.8.4.4.1 No HDFS Resources Reported on a Node

- *Symptom:* No information displayed when executing `qghost -F | grep hdfs`
- *Possible Reasons:*
 1. It may take several minutes until HDFS resource information is reported.
 2. Load sensor has no permission to access HDFS information.
 3. The `env.sh` script in the Hadoop configuration directory has an incorrect path to the Hadoop installation or to the Java platform.
 4. The Hadoop configuration directory contains inconsistent information. Null pointer exceptions from the Hadoop framework are a common problem in that case.
- *Solutions:*
 1. Wait for a few minutes. If still no HDFS resource information appears then check for whether the Hadoop load sensor is running on the respective node. Use the `jps -l` command and look for `com.sun.grid.herd.HerdJsv`. Run this command under the same user under which the load sensor has been started. This is either root or the Univa Grid Engine admin user. If the load sensor is not running then ...

3.8.8.4.4.2 Job Tracker or Task Tracker not Running

3.9 Ensuring High Availability

For an introduction to the shadow master concept see also [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 [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 [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

3.9.6.1 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.

3.9.6.2 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/
```

3.9.6.3 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 <hostname> 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 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 [,É]`
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:

Calendar configuration attributes

Attribute	Value specification
<code>calendar_name</code>	The name of the calendar to be used when attaching it to queues or when administering the calendar definition.
<code>year</code>	<p>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 field is defines as follows:</p> <pre> NONE year_day_range_list = daytime_range_list [= state] year_day_range_list [= daytime_range_list] = state state </pre> <ul style="list-style-type: none"> • <code>NONE</code> means no definition is made on the year basis. • If a definition is made on a yearly basis, at least one of the <code>year_day_range_list</code>, <code>daytime_range_list</code> and <code>state</code> 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 <code>year_day_range_list</code> nor explicitly. <p>and the syntactical components are defined as follows:</p>

	<pre> 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 </pre>
week	<p>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:</p> <pre> 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} ... </pre> <p>Where</p> <ul style="list-style-type: none"> • NONE means, no definition is made on the week basis • if a definition is made on the week basis, at least one of <code>week_day_range_list</code>, <code>daytime_range_list</code> or <code>state</code> always have to be present. • every day in the week is assumed if <code>week_day_range_list</code> is omitted. • syntax and semantics of <code>daytime_range_list</code> and <code>state</code> 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 <code>week_day_range_list</code> nor explicitly <p>and where <code>week_day_range_list</code> is defined as</p> <pre> week_day_range_list := week_day-week_day week_day, É week_day := mon tue wed thu fri sat sun </pre> <p>with <code>week_day_range_list</code> the <code>week_day</code> identifiers must be different.</p>

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,
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,
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,
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 a 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 E
```

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 where 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        *              *              /
```


3.13 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
- ECC 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/loadensors/` 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/loadensors/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):

- `cuda`tools (tested with 4.0.17)
- `cuda`toolkit (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`

3.13.1.1 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)

# global rules: all, clean, depend

all: cuda_load_sensor

clean:
    $(RM) $(CUDA_OBJS) cuda_load_sensor

# rules to build the CUDA load sensor

cuda_load_sensor: $(CUDA_OBJS)
    $(CC) -o cuda_load_sensor $(CUDA_OBJS) $(CUDA_LDFLAGS) $(CUDA_LIBS)

# 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

3.13.2.1 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:

<code>cuda</code>	<code>cuda</code>	<code>INT</code>	<code><=</code>	<code>YES</code>	<code>JOB</code>	<code>0</code>	<code>1000</code>
-------------------	-------------------	------------------	--------------------	------------------	------------------	----------------	-------------------

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
```

3.13.2.2 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 be initialized.

```
qconf -aattr exechost complex_values cuda=1 <hostname>
```

To remove the "cuda" complex value, the following step must be performed:

```
qconf -dattr exechost complex_values cuda=1
```

3.13.2.3 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
```

3.13.2.4 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 Special Tools

3.14.1 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
```

3.14.2 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.

Go back to the [Univa Grid Engine Documentation](#) main page.