AGENDA

1. Introduction
2. Hosts
   - Host Groups
3. Setting up a queue
   - Checkpointing
   - Parallel Environments
   - Load and Suspend Thresholds
   - Limits
   - Subordinates
4. Complex Resource Attributes
   - Consumable resources
   - LoadSensor
5. User Access
   - Configuring User Access
   - Users
   - Projects
   - Department
6. Questions

Jordi Blasco (jordi.blasco@xrqtc.org)  Grid Engine Training (Part 1)
Batch Queue System

is a software application that is in charge of unattended background executions, commonly known for historical reasons as batch processing.a


What is a queue?

- A queue offers a set of resources for a similar jobs.
- The queues use to have some limits to manage the computational resources efficiency.
- Only few BQS have consumable resources control like concurrent licenses limitation.
How it works?

- The users sent their jobs with qsub, with a detailed resources needs (mem., cputime, disk, number of cores, licenses, ...)
- The manager register the job.
- When all the resources are available, the manager send the job to execution nodes following a complex allocation rules (priority, urgency, etc.)
- Some BQS needs more complex scheduler layer like Maui, Moab cluster suite.
- The users can view the job status using qstat.
- The users can delete their job using qdel.
About Grid Engine Project

History

- Previously known as CODINE (COMputing in DIstributed Networked Environments).
- In 2000, Sun acquired Gridware, Inc.
- In 2001, Sun made the source code available
- In 2010, Oracle acquired Sun.
- In response to this, the Grid Engine community started the develop forked versions of GE.
**GRID ENGINE FORKS**

- **UGE** Univa Grid Engine
- **OGE** Oracle Grid Engine
- **SGE** Son of Grid Engine
- **OGS** Open Grid Scheduler

**COMPARING GRID ENGINE**

<table>
<thead>
<tr>
<th></th>
<th>UGE</th>
<th>OGE</th>
<th>SGE</th>
<th>OGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Core</td>
<td>6.2U5</td>
<td>6.2U7</td>
<td>6.2U5</td>
<td>6.2U5</td>
</tr>
<tr>
<td>Current Version</td>
<td>8.0.1</td>
<td>6.2U5p2</td>
<td>8.0.0b*</td>
<td>6.2U5+</td>
</tr>
<tr>
<td>License</td>
<td>Commercial</td>
<td>Commercial</td>
<td>SISSEL</td>
<td>SISSEL</td>
</tr>
<tr>
<td>Support level</td>
<td>Enterprise</td>
<td>Passive</td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>Activity</td>
<td>high</td>
<td>medium</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td>Source Access</td>
<td>yes</td>
<td>yes*</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Public Roadmap</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

*a since 2011-09-29 SGE use Univa Grid Engine 8.0.0 source code core*
Setting up GE with `qmon(GUI)` & `qconf(CLI)`.
# Setting up Grid Engine

## Core admin commands

- **qconf** Admin tool for adding/changing/configuring the Grid Engine system.
- **qstat & qhost** Tools for monitoring.
- **qmod** Modify & disable an existing queue, clear error states, etc.
- **qalter** Change attribute of pending job.
**Most popular syntax**

- Nobody uses (or say that uses) `qmon`.
- Most admins use `qconf -[smA]`.
- Gurus use `qconf -[adm]rattr` and `qconf -purge`. 
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A</td>
<td>a (add)</td>
</tr>
<tr>
<td>-D</td>
<td>d (delete)</td>
</tr>
<tr>
<td>-M</td>
<td>m (modify)</td>
</tr>
<tr>
<td>-r (replace)</td>
<td></td>
</tr>
<tr>
<td>-s (show)</td>
<td></td>
</tr>
<tr>
<td>-purge (removes any overridden settings)</td>
<td></td>
</tr>
</tbody>
</table>
SGE information about global configuration settings.

**Show configuration**
- `qconf -sconf | qconf -sconf global`
- `qconf -sconf <host>`

**Modify configuration**
- `qconf -mconf | qconf -mconf global`
- `qconf -mconf <host>`
Host groups and hosts setting up

There are 5 hosts class

1. Administration Host
2. Submit Host.
3. Host Groups.
4. Execution Host.
5. Master (y shadow)
**Execution Host**

### Setting up execution hosts

1. List all execution hosts `qconf -sel`
2. Modify execution host `qconf -me <hostname>`
3. Delete execution host `qconf -de <hostname>`
4. Show execution host configuration `qconf -se <hostname>`
**EXECUTION HOST**

---

**SETTING UP EXECUTION HOSTS**

1. from GE master: `qconf -ah xhpc01`
2. from execution node: `$_GE_ROOT/install__execd`
Setting up execution hosts

1. Scaling Factors
2. Resource Attributes
3. Access Permissions
4. Reporting Variables
### QCONF -ME XHPC01.XI.XRQTC.ORG

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>xhpc01.xi.xrqtc.org</td>
</tr>
<tr>
<td>load_scaling</td>
<td>NONE</td>
</tr>
<tr>
<td>complex_values</td>
<td>NONE</td>
</tr>
<tr>
<td>user_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>xuser_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>projects</td>
<td>NONE</td>
</tr>
<tr>
<td>xprojects</td>
<td>NONE</td>
</tr>
<tr>
<td>usage_scaling</td>
<td>NONE</td>
</tr>
<tr>
<td>report_variables</td>
<td>NONE</td>
</tr>
</tbody>
</table>
Trick: usage_scaling

If you have multiple architectures, you should use usage_scaling to normalize the usage statistics.

i.e. Intel Westmere X5650 is 5.2 times faster than AMD Opteron 275.

usage_scaling cpu=12.409000
Trick : complex_values

If you need multiple queue instances in the same node, you will need to set up the number of slots as a complex_values. We suggest to use the mem_free complex value to request the needed memory for a particular job.

```
mem_free=47765M, slots=12
```
**Trick: Host Aliases**

It’s very useful if you pretend to use first NIC for Cluster File System and the second one dedicated to MPI (like low latency network).

Default location

$SGE_ROOT/$SGE_CELL/common/host_aliases

Simple format: `<name> <alias to use>`

- xhpc00 192.198.1.10
- xhpc01 192.198.1.11
- xhpc02 192.198.1.12
- xhpc03 192.198.1.13
- xhpc04 192.198.1.14
Setting up host groups

- You can define the host groups with @ character
- The host groups are useful to define a similar nodes.
- You can aggregate by architecture, OS, etc.
- You can set up a new host group using other groups.
- ie: we can define the groups SLES, RHEL and Debian.
- and a new group called Linux that contains the previous groups.

Trick: Infiniband Islands

We suggest you to set up a host group for each infiniband island.
**Setting up Host Groups**

- New hostgroup (interactive) `qconf -ahgrp @<name>`
- New hostgroup (from template file) `qconf -Ahgrp file.dat`
- Modify hostgroup (interactive) `qconf -mhgrp @<name>`
- List all configured hostgroups `qconf -shgrpl`
- Show an existing hostgroup `qconf -shgrp @<name>`
In a heterogeneous scenarios
You can alter this configuration by node or host groups.

With qmon we have to edit 11 tabs.

1. General Configuration
2. Execution Method
3. Checkpointing
4. Parallel Environment
5. Load/Suspend Thresholds
6. Limits
7. Complex
8. Subordinates
9. User Access
10. Project Access
11. Owners
**QCONF -AQ XHPC.Q**

<table>
<thead>
<tr>
<th>Command</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>qconf -sq xhpc.q</td>
<td></td>
</tr>
<tr>
<td>qname</td>
<td>xhpc.q</td>
</tr>
<tr>
<td>hostlist</td>
<td>@xhpc</td>
</tr>
<tr>
<td>seq_no</td>
<td>0</td>
</tr>
<tr>
<td>load_thresholds</td>
<td>np_load_avg=1.75</td>
</tr>
<tr>
<td>suspend_thresholds</td>
<td>NONE</td>
</tr>
<tr>
<td>nsuspend</td>
<td>1</td>
</tr>
<tr>
<td>suspend_interval</td>
<td>00:05:00</td>
</tr>
<tr>
<td>priority</td>
<td>0</td>
</tr>
<tr>
<td>min_cpu_interval</td>
<td>00:05:00</td>
</tr>
<tr>
<td>processors</td>
<td>UNDEFINED</td>
</tr>
<tr>
<td>qtype</td>
<td>BATCH INTERACTIVE</td>
</tr>
<tr>
<td>ckpt_list</td>
<td>NONE</td>
</tr>
<tr>
<td>pe_list</td>
<td>make smp ompi mpi</td>
</tr>
<tr>
<td>rerun</td>
<td>FALSE</td>
</tr>
<tr>
<td>slots</td>
<td>12</td>
</tr>
<tr>
<td>tmpdir</td>
<td>/scratch</td>
</tr>
<tr>
<td>shell</td>
<td>/bin/bash</td>
</tr>
<tr>
<td>prolog</td>
<td>NONE</td>
</tr>
<tr>
<td>epilog</td>
<td>NONE</td>
</tr>
<tr>
<td>shell_start_mode</td>
<td>posix_compliant</td>
</tr>
<tr>
<td>starter_method</td>
<td>NONE</td>
</tr>
<tr>
<td>suspend_method</td>
<td>NONE</td>
</tr>
<tr>
<td>resume_method</td>
<td>NONE</td>
</tr>
<tr>
<td>terminate_method</td>
<td>NONE</td>
</tr>
<tr>
<td>notify</td>
<td>00:00:60</td>
</tr>
<tr>
<td>owner_list</td>
<td>NONE</td>
</tr>
<tr>
<td>user_lists</td>
<td>jb lab</td>
</tr>
<tr>
<td>xuser_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>subordinate_list</td>
<td>NONE</td>
</tr>
<tr>
<td>complex_values</td>
<td>exclusive=true</td>
</tr>
<tr>
<td>projects</td>
<td>NONE</td>
</tr>
<tr>
<td>xprojects</td>
<td>NONE</td>
</tr>
<tr>
<td>calendar</td>
<td>NONE</td>
</tr>
<tr>
<td>initial_state</td>
<td>default</td>
</tr>
<tr>
<td>s_rt</td>
<td>INFINITY</td>
</tr>
<tr>
<td>h_rt</td>
<td>INFINITY</td>
</tr>
<tr>
<td>s_cpu</td>
<td>INFINITY</td>
</tr>
<tr>
<td>h_cpu</td>
<td>INFINITY</td>
</tr>
<tr>
<td>s_fsize</td>
<td>INFINITY</td>
</tr>
<tr>
<td>h_fsize</td>
<td>INFINITY</td>
</tr>
<tr>
<td>s_data</td>
<td>INFINITY</td>
</tr>
<tr>
<td>h_data</td>
<td>INFINITY</td>
</tr>
<tr>
<td>s_stack</td>
<td>INFINITY</td>
</tr>
<tr>
<td>h_stack</td>
<td>INFINITY</td>
</tr>
<tr>
<td>s_core</td>
<td>INFINITY</td>
</tr>
<tr>
<td>h_core</td>
<td>INFINITY</td>
</tr>
<tr>
<td>s_rss</td>
<td>INFINITY</td>
</tr>
<tr>
<td>h_rss</td>
<td>INFINITY</td>
</tr>
<tr>
<td>s_vmem</td>
<td>INFINITY</td>
</tr>
<tr>
<td>h_vmem</td>
<td>INFINITY</td>
</tr>
</tbody>
</table>

**Jordi Blasco** (jordi.blasco@xqtc.org)  
**Grid Engine Training (Part 1)**
## Global Configuration

- **Processors**: A specifier for the processor set to be used by the jobs running in that queue.
- **tmp**: Temporary directory path (/tmp or /scratch)
- **Rerun Jobs**: The queue’s default rerun policy to be enforced on jobs that were aborted, for example, due to system crashes.
- **Notify Time**: The time to wait between delivery of SIGUSR1/SIGUSR2 signals and suspend or kill signals.
- **Slots**: The number of jobs running concurrently in the queue.
- **Type**: Type can be Batch, Interactive, or both.
- **Limits**: Soft and hard limits for cputime, core, disc, memory.
TRICK: HYBRID QUEUES, BATCH AND INTERACTIVE

We suggest to define interactive exceptions inside the queue configuration. In the following example all the interactive requests will be allocated on the xhpc01 node.

```
qtype BATCH,[xhpc01=INTERACTIVE]
```
SETTING UP A QUEUE

**Execution Method**

- **Prolog.** Script that execute something before the job is started.
- **Epilog.** Script that execute something after the job is finished.
- There are other options: Starter Method, Suspend Method, Resume Method, Terminate Method.
Setting up Checkpointing

- MinCpuTime. The periodic checkpoint interval.
- Referenced Ckpt Objects. A list of checkpointing environments associated with the queue.

BLCR - Berkeley Lab Checkpoint/Restart

http://ftg.lbl.gov/CheckpointRestart/CheckpointRestart.shtml

We have a modified version of BLCR and GE integration scripts written by (Liang Peng & Lip Kian) in our GIT server:
http://orion.xrqtc.com/git/?p=blcr_sge_integration/.git;a=summary
## Setting up Parallel Environments

GE provide environment to run parallel jobs:

- Parallel programs over SMP (OpenMP, posix threads, etc.).
- Message-Passing environments (MPI).
- The guide explain how to create a PE for each SO or distribution.
- We suggest to create generic and homogenous PE for all queues.

### TRICK: Modules and Mpirun wrapper

We suggest to add in the modulefiles of each application a wrapper of mpirun command. (more info @ Environment Modules Talk by C.Acosta & D.Masó)

```
set-alias "MPIRUN" "mpirun -np \$NSLOTS --mca btl self,openib \$1"
```
### Setting up Parallel Environments

**Show configuration:**
- Show configuration: `qconf -sp <PE Name>`
- Add new PE: `qconf -ap <PE Name>`
- Add new PE: `qconf -ap <PE Name>`
- Add new PE from file: `qconf -Ap file.dat`
- Modify PE: `qconf -mp <PE Name>`
- List all PE’s: `qconf -spl`
Setting up Parallel Environments

- Name
- Slots
- User List / Xuser Lists
- Start Proc Args
- Stop Proc Args
- Allocacion Rule
- Urgency Slots
- Control Slaves (No, except in special cases)
- Job is first task (PVM)
**Allocation Rules**

- A positive integer fixes the number of processes for each suitable host.

- `$pe_slots`: the full range of processes of a job will be allocated on a single host.

- `$fill_up`: Starting from the best suitable host/queue, all available slots are allocated. Further hosts and queues are ”filled up” as long as a job still requires slots for parallel tasks.

- `$round_robin`: From all suitable hosts a single slot is allocated until all tasks requested by the parallel job are dispatched. If more tasks are requested than suitable hosts are found, allocation starts again from the first host. The allocation scheme walks through suitable hosts in a best-suitable-first order.
Parallel Environments

**SMP** : `qconf -sp smp`

- `pe_name` : smp
- `slots` : 999
- `user_lists` : @allusers
- `xuser_lists` : NONE
- `start_proc_args` : /bin/true
- `stop_proc_args` : /bin/true
- `allocation_rule` : $pe_slots
- `control_slaves` : TRUE
- `job_is_first_task` : FALSE
- `urgency_slots` : min
- `accounting_summary` : FALSE
**OpenMPI**

*OpenMPI* compiled with sge integrations flags (`-with-sge`)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pe_name</code></td>
<td><code>ompi</code></td>
</tr>
<tr>
<td><code>slots</code></td>
<td><code>999</code></td>
</tr>
<tr>
<td><code>user_lists</code></td>
<td><code>@allusers</code></td>
</tr>
<tr>
<td><code>xuser_lists</code></td>
<td><code>NONE</code></td>
</tr>
<tr>
<td><code>start_proc_args</code></td>
<td><code>/bin/true</code></td>
</tr>
<tr>
<td><code>stop_proc_args</code></td>
<td><code>/bin/true</code></td>
</tr>
<tr>
<td><code>allocation_rule</code></td>
<td><code>$round_robin</code></td>
</tr>
<tr>
<td><code>control_slaves</code></td>
<td><code>TRUE</code></td>
</tr>
<tr>
<td><code>job_is_first_task</code></td>
<td><code>FALSE</code></td>
</tr>
<tr>
<td><code>urgency_slots</code></td>
<td><code>min</code></td>
</tr>
<tr>
<td><code>accounting_summary</code></td>
<td><code>FALSE</code></td>
</tr>
</tbody>
</table>
### MPI General: qconf -sp mpi

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe_name</td>
<td>mpi</td>
</tr>
<tr>
<td>slots</td>
<td>999</td>
</tr>
<tr>
<td>user_lists</td>
<td>@allusers</td>
</tr>
<tr>
<td>xuser_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>start_proc_args</td>
<td>/GE/mpi/startmpi.sh -catch_rsh $pe_hostfile</td>
</tr>
<tr>
<td>stop_proc_args</td>
<td>/GE/mpi/stopmpi.sh</td>
</tr>
<tr>
<td>allocation_rule</td>
<td>$fill_up</td>
</tr>
<tr>
<td>control_slaves</td>
<td>TRUE</td>
</tr>
<tr>
<td>job_is_first_task</td>
<td>FALSE</td>
</tr>
<tr>
<td>urgency_slots</td>
<td>min</td>
</tr>
<tr>
<td>accounting_summary</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
### LINDA: qconf -sp LINDA

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe_name</td>
<td>linda</td>
</tr>
<tr>
<td>slots</td>
<td>999</td>
</tr>
<tr>
<td>user_lists</td>
<td>@allusers</td>
</tr>
<tr>
<td>xuser_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>start_proc_args</td>
<td>/GE/mpi/startlinda.sh -catch_rsh $pe_hostfile</td>
</tr>
<tr>
<td>stop_proc_args</td>
<td>/GE/mpi/stoplinda.sh</td>
</tr>
<tr>
<td>allocation_rule</td>
<td>$fill_up</td>
</tr>
<tr>
<td>control_slaves</td>
<td>TRUE</td>
</tr>
<tr>
<td>job_is_first_task</td>
<td>FALSE</td>
</tr>
<tr>
<td>urgency_slots</td>
<td>min</td>
</tr>
<tr>
<td>accounting_summary</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Parallel Environments

JumboMEM : qconf -sp jumbo

- pe_name: jumbo
- slots: 999
- user_lists: iqtc
- xuser_lists: NONE
- start_proc_args: /bin/true
- stop_proc_args: /bin/true
- allocation_rule: 1
- control_slaves: TRUE
- job_is_first_task: FALSE
- urgency_slots: min
- accounting_summary: FALSE
TRICK : INFINIBAND ISLANDS

We suggest to allocate mpi jobs inside the same IB switch to get better performance (IB Islands). You will need to setup a PE for each switch and setup your queue with the exceptions. In the following example we have 2 IB switch and we have setup 2 PE (ompi_1 & ompi_2)

```
qconf -sq xhpc.q | grep pe_list
pe_list               make,[@hostib1=ompi_1 smp],[@hostib2=ompi_2 smp]
```

As a user, you only have to change:
- `pe ompi 72` for `pe "ompi_*" 72`
**Load and Suspend Thresholds**

- **Load Thresholds**: In the case of load thresholds, overload prevents the queue from receiving further jobs. (Default 1.75)

- **Suspend Thresholds**: In the case of suspend thresholds, overload suspends jobs in the queue in order to reduce the load. (Default 4.0)
**Setting up limits**

- GE allow to setting up limits by queue, host or host group.
- The limits prevents potential abuse from users or jobs.
- The limits allows to manage better the current computational resources.

**The limits are useful to define**

- Fast queues with low walltime.
- Queues with huge memory needs.
- Queues with special filesystem occupation needs.
**Limits**

**Setting up limits**

- `s_cpu` per-process CPU time limit in seconds.
- `h_cpu` per-job CPU time limit in seconds.
- `h_data` per-job maximum memory limit in bytes.
- `s_core` per-process maximum core file size in bytes.
- `h_vmem` same as `h_data` (if both are set the minimum is used).
- `s_data` per-process maximum memory limit in bytes.
- `h_fsize` total number of disk blocks that this job can create.
- `s_vmem` same as `s_data` (if both are set the minimum is used).
## Limits

### Example of limits and queues

<table>
<thead>
<tr>
<th>Name</th>
<th>Cores(#)</th>
<th>Mem (GB/core)</th>
<th>Disk(GB)</th>
<th>cputime(Hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>12h</td>
</tr>
<tr>
<td>large</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>240h</td>
</tr>
<tr>
<td>mem</td>
<td>–</td>
<td>2</td>
<td>120</td>
<td>–</td>
</tr>
<tr>
<td>xmem</td>
<td>–</td>
<td>4</td>
<td>220</td>
<td>–</td>
</tr>
<tr>
<td>jumbo</td>
<td>–</td>
<td>12</td>
<td>220</td>
<td>–</td>
</tr>
<tr>
<td>disk</td>
<td>–</td>
<td>–</td>
<td>120</td>
<td>–</td>
</tr>
<tr>
<td>xdisk</td>
<td>–</td>
<td>–</td>
<td>220</td>
<td>–</td>
</tr>
<tr>
<td>bigdisk</td>
<td>–</td>
<td>–</td>
<td>2048</td>
<td>–</td>
</tr>
<tr>
<td>ondemand</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Use the subordinate queue facility to implement high priority and low priority queues as well as standalone queues.

Subordinated queues are suspended if the configured queue becomes busy. (classic view)

Subordinated queues are resumed when the configured queue is no longer busy. (classic view)

Slot-wise preemption let you to suspend some jobs of subordinated queue to run more priority job.
**SUBORDINATE QUEUES**

- **Queue.** A list of the queues that are subordinated to the configured queue.
- **Max Slots.** For any subordinated queue, you can configure the number of job slots that must be filled in the configured queue to trigger a suspension. If no maximum slot value is specified, all job slots must be filled to trigger suspension of the corresponding queue.
**TRICK: SLOT-WISE PREEMPTION**

We suggest to subordinate queues with large walltime limits. The experience tells that the short jobs may be more priority, and long term jobs will not note difference if this jobs is suspended for a short period. 

- **sr**: means to suspend the shortest running job
- **lr**: means to suspend the longest running job.

**Ex1 : Setting up (seq. number)**

```bash
# qconf -sq short.q | grep subordinate_list
subordinate_list slots=24(large.q:1:sr,jumbo.q:2:lr)
# qconf -sq large.q | grep subordinate_list
subordinate_list NONE
# qconf -sq jumbo.q | grep subordinate_list
subordinate_list NONE
```
Ex2: Setting up (tree dependency)

# qconf -sq short.q | grep subordinate_list
subordinate_list slots=24(large.q:0:sr)

# qconf -sq large.q | grep subordinate_list
subordinate_list slots=12(jumbo.q:0:sr)

# qconf -sq jumbo.q | grep subordinate_list
subordinate_list NONE
**Complex Resource Attributes (CRA)**

**You can find 3 main types**
- Static (like the hostname, architecture, etc.)
- Consumable (like free memory, available license, ...)
- Measured (server load, idle time, etc.)

**The CRA can be defined by:**
- queue
- host
- host group
- global
**Complex Resource Attributes (CRA)**

- The most common usage is to get control over:
  - Available free memory.
  - Available software license.
  - Available disk free

**Setting up CRA**

- Show configuration `qconf -sc`
- Modify configuration `qconf -mc`
**Complex Resource Attributes**

**Setting up CRA**

- Name of the attribute
- Shortcut
- Type (STRING, INT, TIME,...)
- Relational operator used by the scheduler
- Requestable flag.
- Consumable flag.
- Default request value.
- Urgency value.
**Ex. Free memory control**

- It can minimize the swap usage.
- It guarantees free memory space for requested jobs.
- It prevents potential memory abuse.

**Setting up**

- Modify `virtual_free` in ”Complex Configuration”.
- Now `virtual_free` has to be a consumable resource.
- Inside the node or queue configuration, you must set the maximum value.
Ex. Exclusive usage

- It guarantees all node computational resources.
- It prevents potential abuse of others jobs running in the same node.
- The jobs will be sent with `-l excl=true` flag.

Setting up

```
qconf -sc | grep exclusive
exclusive excl BOOL EXCL YES YES 0
qconf -se xhpc01
...
complex_values exclusive=true
...
```
**TRICK: free slots**

If you want to manage multiple queue instances on the same node you will need to setup the **slots** as a consumable resource.

**SETTING UP**

```
qconf -aattr exechost complex_values slots=12 xhpc01
```
MULTIPLE QUEUE INSTANCES
TRICK : TOPOLOGY-AWARE SCHEDULING

Some applications have serious resource needs, and these don’t play well with others on the same node. For these applications, how the threads are distributed across the cluster makes a huge difference (fighting over cache space or I/O bandwidth). Most operating systems don’t make sense of core affinity (Maybe only Solaris, do it). With topology-aware scheduling (release 6.2.U5), the users are allowed to specify three different flavors of distribution strategy:

- **linear**: the execution daemon will place the job’s threads/processes on consecutive cores if possible.
- **striding**: tells the execution daemon to place the job on every #th core, (e.g. every 1th core)
- **explicit**: lets the user decide exactly which cores will be assigned to the job.

The core binding is a request, not a requirement. New default complex will be needed (m_core m_socket).
**Consumable Resources**

### E.G Linear

```
qsub -binding linear:6 -l m_core=12 -l m_socket=2 job.sh
```

### Striding

```
qsub -binding striding:2:4 -l m_core=8 -l m_socket=2 job.sh
```

### Explicit

```
qsub -binding explicit:0,0:0,3:1,0:1,3 -l m_core=8 -l m_socket=2 job.sh
```
**Control of Resource Occupation**

Grid Engine allows to integrate your own scripts to check and control the current resource occupation.

**Output Format**

```
begin
global:resource:N
end
```
LoadSensor - FlexLM

If you need to get control about available float license, we suggest to follow the integration of the scripts written by Dr. Mark Olesen.

http://wiki.gridengine.info/wiki/index.php/Olesen-FLEXlm-Integration
There are 4 classes of users:

- **Managers**: Can control any aspect of grid engine (qconf -am <user>)
- **Operators**: Same as manager but no ability to add, delete or change a queue configuration (qconf -ao <user>)
- **Owners**: Suspend, resume and disable one or more queues that a user may 'own' ("owner" in Queue conf)
- **Users**: Can use and query system but can’t change anything
User Access Lists

Userset

The userset is a group of users, that can be created by usernames or user groups.
It can be "Access Lists", "Departments" and "Projects".

- Show userset list: `qconf -sul`
- Show userset content: `qconf -su <userset>`

Department/Access List

Department or access list? One user can belong only of a department, but the same user can be on several access list.
### QCONF SYNTAX (Users)

- Show user list `qconf -suserl`
- Add new user `qconf -auser <name>`
- Show user configuration `qconf -suser <name>`
- Delete user `qconf -duser <name>`
- Modify user configuration `qconf -muser <name>`

```sh
# qconf -auser hpckp
# qconf -suser hpckp
name hpckp
oticket 0
fshare 0
delete_time 0
default_project lab
```
TRICK: Massive user creation with LDAP

```
ldapsearch -x | grep uid= | sed -e "s/dn: uid=/ /g" | sed -e "s/,ou=/ /g" | cut -d, -f1 > userlist.dat
```
**TRICK : Massive user creation with LDAP**

Creating a template "auser.dat".

```
name template  
oticket 0     
fshare 0      
delete_time 0 
default_project NONE
```
**TRICK : Massive user creation with LDAP script**

```bash
for i in $(cat userlist.dat | gawk '{print $1}')
do
g=$(grep $i userlist.dat | gawk '{print $2}')
cat auser.dat | sed -e "s/template/$i/g" |
    sed -e "s/NONE/$g/g" > /tmp/$i.dat
qconf -Auser /tmp/$i.dat
qconf -au $i $g
rm /tmp/$i.dat
done
```
Projects

Projects objects

- Show project list `qconf -sprj`
- Add new project `qconf -aprj <name>`
- Show project configuration `qconf -sprj <name>`
- Delete project `qconf -dprj <name>`
- Modify project configuration `qconf -mprj <name>`

```bash
# qconf -aprj lab
# qconf -sprj lab
name lab
oticket 100000  <---- very usefull for admins
fshare 0
acl lab
xacl NONE
```
Department

Department Objects

- Show department list `qconf -sul`
- Add new department `qconf -au <name>`
- Show department configuration `qconf -su <name>`
- Delete department `qconf -du <name>`
- Modify department configuration `qconf -mu <name>`

```bash
# qconf -au lab
# qconf -su lab
name  lab
type  ACL
fshare 0
oticket 0
entries hpckp
```
**REFERENCES**

- Sun Grid Engine Installation Guide
- Sun Grid Engine Administrator Guide
- Sun Grid Engine User Guide
- http://bioteam.net
- http://www.univa.com
- https://arc.liv.ac.uk/trac/SGE
- http://gridengine.org
- http://gridscheduler.sourceforge.net
- http://gridengine.info
AGENDA

1 SCHEDULER
   - Scheduler Strategies
   - Dynamic Resource Management
   - Tickets
   - Queue Sorting
   - Job Sorting

2 POLICIES
   - Urgency policy
   - Ticket Policy Hierarchy
   - Override policy
   - Functional policy
   - Share-based policy

3 RESOURCE QUOTAS

4 RESOURCE RESERVATION

5 BACKUP AND RESTORE

6 WHAT IS MISSING?

7 QUESTIONS
SGE schedules jobs based on the following criteria:

- The cluster’s current load
- The jobs’ relative importance
- The hosts’ relative performance
- The jobs’ resource requirements (CPU, memory, and I/O bandwidth)
Scheduling Strategies

- Dynamic resource management (CPU share).
- Queue sorting.
- Job sorting.
- Resource reservation and backfilling.
SGE uses a **weighted combination** of the following three ticket-based policies to implement automated job scheduling strategies:

- Share-based
- Functional (sometimes called Priority)
- Override
SCHEDULING

 TICKETS

- Each policy has a pool of tickets.
- Each policy allocates some tickets to each new job.
- If some policy don’t have more tickets, this will not be used.
- If two or more policies have an equal number of tickets, then this policies have equal weight.
### Queue Sorting

SGE attempts to fill up queues using the following factors:

1. Load reporting.
2. Load scaling.
3. Load adjustment.
4. Sequence number.
Job Sorting

By default, the order is (first-in-first-out - FIFO). The administrator has the following means to control the job order:

- Ticket-based job priority.
- Urgency-based job priority.
- POSIX priority. Range of priorities from -1023 to 1024. The default is 0 (ex. qsub -p 1024 ...).
- Maximum number of user or user group jobs.
SCHEDULING

**JOB PRIORITY**

The following formula expresses how a job’s priority values are determined:

\[ \eta_{\text{JobPriority}} = \eta_{\text{Urgency}} \cdot \omega_{\text{Urgency}} + \eta_{\text{Ticket}} \cdot \omega_{\text{Ticket}} + \eta_{\text{Priority}} \cdot \omega_{\text{Priority}} \]

**TRICK : NORMALIZE**

We suggest to normalize the sumatory of all weight contributions:

\[ \sum_i \omega_i = 1. \]

\[ \omega_{\text{Priority}} = 0.01 \]
\[ \omega_{\text{Urgency}} = 0.1 \]
\[ \omega_{\text{Ticket}} = 0.89 \]
As administrator, you can define high-level usage policies that are customized for your site. Four such policies are available:

- **Urgency policy** (resource availability + Waiting)
- **Share-based policy** (usage + fair share)
- **Functional policy** (fair share)
- **Override policy** (perfect for us: SysAdmins :-) )

In the following slides we are going to focus on Urgency policy and Share-based policy.
### QCONF - SSCONF

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>algorithm</strong></td>
<td>default</td>
</tr>
<tr>
<td><strong>schedule_interval</strong></td>
<td>0:0:7</td>
</tr>
<tr>
<td><strong>maxujobs</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>queue_sort_method</strong></td>
<td>load</td>
</tr>
<tr>
<td><strong>job_load_adjustments</strong></td>
<td>np_load_avg=0.50 share_functional_shares</td>
</tr>
<tr>
<td><strong>load_adjustment_decay_time</strong></td>
<td>0:7:30</td>
</tr>
<tr>
<td><strong>load_formula</strong></td>
<td>np_load_avg</td>
</tr>
<tr>
<td><strong>schedd_job_info</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>flush_submit_sec</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>flush_finish_sec</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>params</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>reprioritize_interval</strong></td>
<td>0:0:0</td>
</tr>
<tr>
<td><strong>halftime</strong></td>
<td>168</td>
</tr>
<tr>
<td><strong>usage_weight_list</strong></td>
<td>cpu=1.,mem=0.,io=0.,weight_urgency</td>
</tr>
<tr>
<td><strong>compensation_factor</strong></td>
<td>2.000000</td>
</tr>
<tr>
<td><strong>weight_user</strong></td>
<td>0.250000</td>
</tr>
<tr>
<td><strong>weight_project</strong></td>
<td>0.250000</td>
</tr>
<tr>
<td><strong>weight_department</strong></td>
<td>0.250000</td>
</tr>
<tr>
<td><strong>weight_job</strong></td>
<td>0.250000</td>
</tr>
<tr>
<td><strong>weight_tickets_functional</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>weight_tickets_share</strong></td>
<td>1000000</td>
</tr>
<tr>
<td><strong>share_override_tickets</strong></td>
<td>TRUE</td>
</tr>
<tr>
<td><strong>share_functional_shares</strong></td>
<td>FALSE</td>
</tr>
<tr>
<td><strong>max_functional_jobs_to_schedule</strong></td>
<td>200</td>
</tr>
<tr>
<td><strong>report_pjob_tickets</strong></td>
<td>TRUE</td>
</tr>
<tr>
<td><strong>max_pending_tasks_per_job</strong></td>
<td>50</td>
</tr>
<tr>
<td><strong>halflife_decay_list</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>policy_hierarchy</strong></td>
<td>OS</td>
</tr>
<tr>
<td><strong>weight_ticket</strong></td>
<td>0.890000</td>
</tr>
<tr>
<td><strong>weight_waiting_time</strong></td>
<td>0.000000</td>
</tr>
<tr>
<td><strong>weight_deadline</strong></td>
<td>3600000.0000.000000</td>
</tr>
<tr>
<td><strong>weight_urgency</strong></td>
<td>0.100000</td>
</tr>
<tr>
<td><strong>weight_priority</strong></td>
<td>0.010000</td>
</tr>
<tr>
<td><strong>max_reservation</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>default_duration</strong></td>
<td>INFINITY</td>
</tr>
</tbody>
</table>

**Questions**

- Urgency policy
- Ticket Policy Hierarchy
- Override policy
- Functional policy
- Share-based policy

**Scheduler Policies**

**Resource Quotas**

**Resource Reservation**

**Backup and restore**

**What is missing?**

**Jordi Blasco (jordi.blasco@xrqtc.org)**

**GridEngine Training (Part 2)**
Urgency Policy

Setting up the Urgency Policy

The Urgency Policy defines an urgency value for each job. This urgency value is determined by the sum of the following three contributing elements:

- Resource requirement contribution (Complex resource).
- Waiting time contribution (seconds).
- Deadline contribution (seconds).
Setting up the Ticket Policy Hierarchy

The Ticket Policy Hierarchy can be a combination of up to three letters. These letters are the first letters of the names of the following three ticket policies:

- S - Share-based
- F - Functional
- O - Override

The following form is recommended for policy_hierarchy settings: [O][S|F]
Override policy

The administrator assigns tickets to the different members of the override categories (users, projects, departments, or jobs). Consequently, the number of tickets that are assigned to a category member determines how many tickets are assigned to jobs under that category member.

- set share_override_tickets=TRUE
- assign otickets to User, project, department or job.
- you can do it with qalter on pending jobs
The functional policy defines entitlement shares for the functional categories. The functional policy can be explained as a simple case of two-level share tree policy. A job can be associated with several categories at the same time. The job belongs to a particular user, for instance, but the job can also belong to a project, a department, and a job class.

- The shares that are defined for its corresponding category member (for example, its project)
- The shares that are given to the category (project instead of user, department, and so on)
## Functional policy

### A very simple way to setting up a Functional policy

Edit SGE configuration (qconf -mconf)
- `enforce_user auto`
- `auto_user_fshare 100`

Edit SGE Scheduler (qconf -msconf)
- `weight_tickets_functional 10000`
- `policy_hierarchy OF`
SHARE-BASED POLICY

SHARE-BASED SCHEDULING

- Share-based policy is very similar to functional policy, but using accumulated usage.
- The unused share proportions are still available for pending jobs associated with other share-tree branches.
- The projects, or users with less accumulated past usage will have more priority.
- This usage is adjusted by a decay factor. "Old" usage has less impact.
- Doing so ensures that all users get close to their fair share of the system during the accumulation period.
- Half-life is how fast the GE "forgets" about a user’s resource consumption.
- The compensation factor enables to limit how much a user or a project can dominate the resources in the near term. (2-10).
Share-based policy

1. Share Tree Policy

- Project A does not need its full allocation of resources
- Project A wants its resources back
- Project A receives compensation for resource usage by Project B
- Usage by Project A and Project B returns to policy assignment

Approximately 50% of Resources

Actual Usage

Assigned Usage

Project A

Project B

Time

source: gridengine.info
SHARE-BASED POLICY

Configuring the Share-Tree Policy

- Identifier.
- Shares.
- Level Percentage.
- Total Percentage.
- Current Resource Usage.
- Targeted Resource Usage. (solo los nodos activos)
- Combined Usage. (permite ajustar por CPU, MEM, I/O)
SHARE-BASED POLICY

LEAF NODE

- All nodes have a unique path in share tree.
- A project is not referenced more than once in share tree.
- A user appears only once in a project subtree.
- A user appears only once outside of a project subtree.
- All leaf nodes in a project subtree reference a known user or the reserved name.
- Project subtrees do not have subprojects.
- If all the users in the same project has the same share (use user default).
SHARE-BASED POLICY

SETTING UP SHARE-BASED POLICY

Edit SGE Scheduler (qconf -msconf)

- weight_tickets_share 1000000
- policy_hierarchy OS
- halftime 168
- compensation_factor 2.000000
Share-based Policy

Setting up Share-tree

The share use to be the number of cores or the % of the whole computational resources.
To be right, we suggest to calculate the share using an scale factor to compensate the relative performance between different family of processors.

- Project lab (10)
- Project QFA (40)
- Project QFB (50)
- User jordi of Project QFA has 30
- User default of Project QFA has 70 (but QFA has 11 users).
Share-based policy

Setting up Share-Tree

```bash
# qconf -sstree
id=0
name=Root
type=0
shares=1
childnodes=1
id=1
name=qfa
type=0
shares=40
childnodes=2,3
id=2
name=default
type=0
shares=70
childnodes=NONE
id=3
name=jordi
type=0
shares=30
childnodes=NONE
```
The Resource Quotas allows to create an ACL’s like:

- `max_u_jobs` on a per-host basis
- Max jobs per user on a per-queue basis
- Per user slot limits on parallel environments
The syntax is very simple and similar to routing rules. The rules can contain:

- Wildcard (*)
- Logical not operator (!)
- Brackets ()
EX: qconf -srqs maxslots

```
{
    name       maxslots
    description "Max. slots per user and project"
    enabled    TRUE
    limit      users {*} projects {exiqc} to slots=48
    limit      users {dani} projects {iqc} to slots=144
    limit      users {*} projects {iqc} to slots=120
    limit      projects {exiqc} to slots=30
    limit      projects {guest} to slots=54
    limit      queues {llarga8d.q} to slots=156
}
```
Resource Reservation

Resource reservation enables you to reserve system resources for specified pending jobs. Those resources are blocked from being used by jobs with lower priority.

Backfill Scheduling

Enables a lower-priority (and small walltime) job to use resources that are blocked due to a resource reservation.
TRICK: Potential performance degradation

Because resource reservation causes the scheduler to look ahead, using resource reservation affects system performance. This performance degradation will be more significant with many pending jobs. To minimize, we suggest to add global consumable "slots=1" and use in sched_conf (qconf -msconf)

```
params
max_reservation
default_duration
```

```
MONITOR=true
100 <---- less is better
INFINITY
```

Check via $SGE_ROOT/$SGE_CELL/default/common/schedule that neither parallel nor sequential jobs get a RESERVE entry.
TRICK: Backup and Restore

The following commands are the best way to backup and restore the SGE configuration.

- `inst_sge -bup`
- `inst_sge -rst`
What is missing?

**Missing features on main project code**

- Accounting web interface (S-GAE - RDLab)
- Monitoring web interface (PHPQstat - XRQTC)
- GPU Integration (Jose Alcantara scripts - XRQTC)
- Power Control (Green Scheduler - XRQTC)
- Efficiency Control (Green Scheduler next release)
- Dynamic quota (new project or contribution?)

Jordi Blasco (jordi.blasco@xrqtc.org)  
GridEngine Training (Part 2)
What is missing?

Questions

Jordi Blasco (jordi.blasco@xrqtc.org) GridEngine Training (Part 2)
REFERENCES

- Sun Grid Engine Guides
- http://bioteam.net
- http://www.univa.com
- https://arc.liv.ac.uk/trac/SGE
- http://gridengine.org
- http://gridscheduler.sourceforge.net
- http://gridengine.info
- http://wikis.sun.com/display/GridEngine/Home
### Requirements
- Xen paravirtualized image set:
  - /xen/vm_hpc00_root
  - /xen/vm_hpc01_root
- OpenSSH Client
- OpenMotif
- Grid Engine Binary files

### ToDo
- ssh -X user@hpc00.xi.xrqtc.org
- Download the GE from the website
- Create $SGE_ROOT directory (/sgf)
- Create gevent and change owner
- Execute ./install_qmaster and follow the instructions
- Add new hosts with: qconf -ah
- Add the submit host (hpc00) with qconf -as hpc00
- Login at hpc01 node
- Create gevent
- Execute: ./install_execd
- Create simbolic links of settings.*sh
- Shutdown and clone the hpc01 to hpc02
- Create a host group
- Create the PEs mpn, omni and mpi.
- Create the queues xhpc,q, fast.q and ondemand,q
- fast.q will have 1h of cputime
- OnDemand.q Initial State = Disabled.
- All nodes has slots=12
- Create a user1, user2 and user3 and dept1, dept2 and dept3
- Create new project of this group in GE
- In Policy configuration:
  - Activate Share-based y Urgency policy
  - In Share Tree Policy:
    - Shares : Sdept01=Sdept02=25, Sdept03=50
    - Compensation Factor = 3.
    - Half-Life Factor = 1 year.
    - Establish new slot’s quota for each group
    - Check the correct usage.

### Quick Reference
- Jobs examples: $SGE_ROOT/examples/jobs/
- qacct Extract accounting information from cluster
- qalter Changes the attributes of submitted but pending jobs
- qconf GE’s cluster, queue etc configuration
- qdel Job deletion
- qhold Holds back submitted jobs for execution
- qhost Shows status information about SGE hosts
- qmod Modify queue statues: enabled or suspended
- qselect List queue matching selection criteria
- qstat Status listing of jobs and queues
- qsub Commandline interface to submit jobs to GE.
- qlogin, qtcsh, qsh - extended command shells that can transparently distribute execution of programs/applications to least loaded hosts via GE.

### Resource Quota Set
```
qconf -srqs
{
    name     maxslots
    description  "Max slots = total_cores*share*2"
    enabled    TRUE
    limit     projects {dept1} to slots=12
    limit     projects {dept2} to slots=12
    limit     projects {dept3} to slots=24
}
```

### Reference
- GE Training Slides
- SGE guides [http://gridengine.sunsource.net](http://gridengine.sunsource.net)


---

**October 2011 - XRQTC ( Xarxa de Referència en Química Teòrica i Computacional )
email: jordi.blasco@xrqtc.org - web: www.xrqtc.org - HPC Knowledge Portal : hpc.xrqtc.org**