Grid Engine Administration

Configuration
This module covers

- Command line operation
- Host & hostgroup configuration
- ‘Cluster’ configuration
- Queue configuration
- Parallel Environments
- Resources & SGE Complex
- Load Sensors
- User access
Command-line Operation
Small number of binaries, but …

- ... many arguments and options
- Must read the manpage for these commands to appreciate them

- qconf
- qstat
- qmod
- qalter
- qdel
- ...
- qsub|qrsh
Core admin commands

- **qconf**
  - Primary admin tool for adding/changing/configuring just about everything in a Grid Engine system

- **qstat | qhost**
  - Primary tools for monitoring

- **qmod**
  - Modify/disable an existing queue, clear error states …

- **qalter**
  - Change attribute of pending job
Advanced Admin Syntax …

- Nobody uses ‘qmon’
- Most admins use “qconf -[smA]”
- Hardcore admins use “qconf -[adm]rattr”
- Wizards use “qconf -purge”
SGE Command Meta Syntax

- **Template Driven**
  - *Add* from file based template
    - `-A<cmd>  <param1>  <param2>  ...`
  - *Delete* from file based template
    - `-D<cmd>  <param1>  <param2>  ...`
  - *Modify* from file based template
    - `-M<cmd>  <param1>  <param2>  ...`

- **Example**
  - *Add a new parallel environment from file*
    - `qconf -Ap ./my-predefined-parallel-environment.txt`
SGE Command Meta Syntax

- Interactive Use
  - Add something
    - `-a<cmd> <param>`
  - Delete something
    - `-d<cmd> <param>`
  - Modify something
    - `-m<cmd> <param>`
  - Show something
    - `-s<cmd> <param>`

- Example
  - Add a new parallel environment interactively
    - `qconf -ap`
    - ... SGE will then open an interactive editor session
      - SGE uses ‘vi’ or whatever is defined by $EDITOR
    - Assuming no syntax error, changes are instantly made live
Meta Syntax Summary

- **-A|a (add)**
- **-D|d (delete)**
- **-M|m (modify)**
- **-s (show)**

- **Capitalized argument means ‘read in from file’**
- **Lowercase means ‘do it interactively’**
  - All SGE commands generally follow this structure
  - Read manpage for ‘qconf’ to see this in action
Configuring Hosts & Hostgroups
Hardcore: qconf -[admr]attr

- Non interactive, very scriptable
- Add, delete, modify, replace
- Primarily for list attributes
Hardcore: qconf -[admr]attr

- Syntax
  - -[admr]attr obj_name attr_name value=[v] obj_id_lst

- “Add host, node1, to hostgroup @allhosts”
  - qconf -aattr hostgroup hostlist node1 @allhosts

- “Change np_load_avg to 2 in load_thresholds in the all.q cluster queue”
  - qconf -mattr queue load_thresholds np_load_avg=2 all.q
Hardcore: Modify vs Replace

- -mattr changes the value of a setting
- -rattr replaces the entire list of settings

Thought Exercise
- Assume:
  - load_thresholds np_load_avg=2, mem_used=2G

What is the effect of:
- qconf -mattr queue load_thresholds np_load_avg=3 all.q
- qconf -rattr queue load_thresholds np_load_avg=2 all.q
Hardcore: Modify vs Replace

- Thought Exercise Solution (-mattr)
- Command:
  - qconf -mattr queue load_thresholds
    np_load_avg=2 all.q
- Result:
  - load_thresholds np_load_avg=2, mem_used=2G
Hardcore: Modify vs Replace

- Thought Exercise Solution (-rattr)
- Command:
  - `qconf -rattr queue load_thresholds np_load_avg=2 all.q`
- Result:
  - `load_thresholds np_load_avg=2`
Hardcore: Replace vs. Purge

- Replace (-rattr) is for list attributes
  - Any attribute, not limited in scope to queues
- Purge (-purge) ONLY for queue instances
  - Removes any overridden settings
  - Example
    - “Remove host-specific slots settings for node01 in all.q …”
    - `qconf -purge queue slots all.q@node01`
Class Exercise:

- Create a new PE object called “dummy”
- Do all of the following without using ‘qmon’ or the ‘qconf -[Am]’ syntax …

1. Create a PE called dummy (via qconf -ap)
2. Add dummy to all.q
3. Remove make from all.q
4. Make make the only PE for all.q
5. Change load_thresholds setting to “np_load_avg=4”
6. Blow away all slot settings from all.q for any single queue instance
7. Extra: Add a slots setting for all.q for any single queue instance
Lab Solution

1. `qconf -aattr queue pe_list dummy all.q`
2. `qconf -dattr queue pe_list dummy all.q`
3. `qconf -rattr queue pe_list make all.q`
4. `qconf -mattr queue load_thresholds np_load_avg=4 all.q`
5. `qconf -purge queue slots all.q@node`
6. Bonus:
   - `qconf -aattr queue slots '[node01=4]' all.q`
Hostgroups

- Convenient way to group hosts
- Hostgroup names must start with “@”
  - @allhosts
  - @bigMemoryhosts
  - @1024CPUhosts
- Hostgroup objects can be used …
  - Queue configurations, access control lists, qsub arguments, etc.

  qsub -q all.q@@bigMemoryhosts ./myjob.sh
Some host group commands

- **New hostgroup (interactive)**
  - `qconf -ahgrp @<name>`

- **New hostgroup (from template file)**
  - `qconf -Ahgrp ./my-predefined-hostgroup.txt`

- **Modify hostgroup (interactive)**
  - `qconf -mhgrp @<name>`

- **List all configured hostgroups**
  - `qconf -shgrpl`

- **Show an existing hostgroup**
  - `qconf -shgrp @<name>`
    - Example: `qconf -shgrp @allhosts`
Related: Reserved hostnames

- These hostnames can not be used within a Grid Engine system:
  - global
  - template
  - all
  - default
  - unknown
  - none
Some host configuration commands

- **List all execution hosts**
  - `qconf -sel`

- **Modify execution host**
  - `qconf -me <hostname>`

- **Delete execution host**
  - `qconf -de <hostname>`

- **Show execution host configuration**
  - `qconf -se <hostname>`
```bash
dag$ qconf -se chrisdag-aliased
hostname               dag-static
load_scaling           NONE
complex_values         NONE
load_values            arch=darwin-x86,num_proc=2,mem_total=4096.000000M, \
                        swap_total=0.000000M,virtual_total=4096.000000M, \
                        load_avg=0.558594,load_short=0.344238, \ 
                        load_medium=0.558594,load_long=0.482910, \ 
                        mem_free=119.156250M,swap_free=0.000000M, \ 
                        virtual_free=119.156250M,mem_used=3976.843750M, \ 
                        swap_used=0.000000M,virtual_used=3976.843750M, \ 
                        cpu=19.300000,np_load_avg=0.279297, \ 
                        np_load_short=0.172119,np_load_medium=0.279297, \ 
                        np_load_long=0.241455
processors            2
user_lists            NONE
xuser_lists           NONE
projects              NONE
xprojects             NONE
usage_scaling         NONE
report_variables      NONE
```
Host configuration parameters

- **hostname**
  - Hostname as SGE understands it

- **load_scaling**
  - Comma separated list of scaling factors to be applied to load values being reported by the sge_execd. Format: `<load value>=<multiplier>, ..`

- **complex_values**
  - Comma separated list. Sets value of host-managed resource attributes. Compared against available consumable resources listed in the SGE complex
    - For consumable resources, this can set a “quota” on new jobs. If the sum of resources consumed by running tasks exceeds a value defined here, no jobs can be placed
    - For non consumable resources, simple relop comparison occurs between job requests, SGE complex and the value reported here. If “true”, job can land on this host.
Host configuration parameters

- **load_values & processors**
  - *Can't be changed here, included so that “qconf -se” shows them*

- **usage_scaling**
  - Same format as load_scaling. Usefulness unknown. Only currently works with “mem=“ and “cpu=“.

- **user_lists & xuser_lists**
  - Comma separated list of named access lists defining who can and cannot make use of this host. If user is listed in both places, access will be denied.

- **projects & xprojects**
  - Same behavior as user lists applied to project membership

- **report_variables**
  - If reporting file is enabled, report this comma separate list of values into it. Settings here will override anything done at a global level
SGE “Cluster” Configuration
Cluster configuration

- “Cluster” means:
  - SGE information about site dependencies and configuration settings

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

- Modify / Edit
  - `qconf -mconf | qconf -sconf global`
  - `qconf -mconf <host>`
SGE ‘Cluster’ Config Params

```
# qconf -sconf

global:
execd_spool_dir /opt/sge61/default/spool
mailer /usr/bin/mail
xterm /usr/X11R6/bin/xterm
load_sensor none
prolog none
epilog none
shell_start_mode posix_compliant
login_shells sh,ksh,csh,tcsh
min_uid 0
min_gid 0
user_lists none
xuser_lists none
projects none
xprojects none
enforce_project false
enforce_user auto
load_report_time 00:00:40
max_unheard 00:05:00
reschedule_unknown 00:00:00
loglevel log_warning
administrator_mail none
set_token_cmd none
pag_cmd none

token_extend_time none
shepherd_cmd none
qmaster_params none
execd_params none
reporting_params
    accounting=true reporting=false \ 
    flush_time=00:00:15 joblog=false \ 
    sharelog=00:00:00
finished_jobs 100
gid_range 20000-20100
qlogin_command /usr/libexec/telnetd
qlogin_daemon /usr/libexec/telnetd
rlogin_daemon /usr/libexec/rlogind
max_aj_instances 2000
max_aj_tasks 75000
max_u_jobs 0
max_jobs 0
auto_user_oticket 0
auto_user_fshare 0
auto_user_default_project none
auto_user_delete_time 86400
delegated_file_staging false
Reprioritize 0
```
Host ‘cluster’ settings

# qconf -sconf chrisdag-aliased
chrisdag-aliased:
mailer                       /usr/bin/mail
xterm                        /usr/X11R6/bin/xterm
qlogin_daemon                /usr/libexec/telnetd
rlogin_daemon                /usr/libexec/rlogind
Some cluster configuration parameters

- Full explanation of all parameters
  - sge_conf (5) man page

- load_sensor
  - Path to script for reporting custom load values, if configured here script will be run on ALL hosts in the cluster

- prolog & epilog
  - Global scripts that can be invoked before|after any job

- shell_start_mode
  - posix_compliant
    - POSIX batch standard says that systems must ignore first line of all scripts in favor of globally configured shell or user configured shell (“qsub -S /bin/csh …”)
  - unix_behavior
    - Honor the environment defined by 1st line in a job script
Some cluster configuration parameters

- **reschedule_unknown**
  - Time to wait after a host enters ‘unknown’ state before rescheduling a job elsewhere
  - Lots of caveats, read the manpage …

- **max_unheard**
  - Mark queue instance in “u” state when no communication received within this interval. Docs say default is “00:2:30” but it may actually be “00:5:00”

- **loglevel = log_err | log_warn | log_info**
  - Adjusts detail/verbosity of the various messages files
  - Useful for debugging and troubleshooting, Default level is “log_info”

- **max_u_jobs & max_jobs**
  - “big stick” approach. Sets global limits on how many jobs can be in the system at one time.

- **qmaster_params, execd_params, reporting_params**
  - Check the sge_conf (5) man page, lots of good stuff can be configured here
More on shell_start_mode

- unix_behavior
  - Scripts: honor the “#!” line of jobscript
  - Binaries: honor the shell named by the queues shell attribute

- posix_compliant
  - Scripts & Binaries: always use queue shell attribute
  - Note: overridden by “-S <shell>” argument or embedded qsub option

- script_from_stdin
  - While still root, read in script
  - Feed script to shell via STDIN
  - Honor the queue shell attribute
### shell_start_mode behavior

<table>
<thead>
<tr>
<th>unix_behavior</th>
<th>Script</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Shell named by #! line of script</td>
<td>Shell named by queue's shell attribute</td>
</tr>
<tr>
<td>posix_compliant</td>
<td>Shell named by queue's shell attribute</td>
<td>Shell named by queue's shell attribute</td>
</tr>
<tr>
<td>script_from_stdin</td>
<td>Shell named by queue's shell attribute</td>
<td>Shell named by queue's shell attribute*</td>
</tr>
</tbody>
</table>

*script_from_stdin is ignored

---

Credit: Dan Templeton
shell_start_mode: Override

- Passing the ‘-S <shell>’ argument will override default shell selection when:
  - posix_compliant or script_from_stdin
- Ultimate Override
  - Configure a custom starter_method
    - Bypasses all other job launching hooks
    - Runs as job owner
    - Arbitrary script used to invoke the job
More on Prolog/Epilog

- Same starter rules as job
  - Except shell_start_mode always “unix_behavior”
- Has same ENV as job context
- Started by sge_shepherd
  - Runs under UID of job submitter
  - “Bookends” PE start/stop methods
- Which gets precedence?
  - Queue overrides Host overrides Global
Remember this?

Hierarchy:
- sge_execd
  - execd owner
  - job owner
- sge_shepherd
  - starter
  - prolog
  - PE start
  - job
  - PE stop
  - epilog

Time:
- starter
- starter
- starter
- starter
- starter

SGE training, consulting and special projects - BioTeam Inc. - http://www.bioteam.net
Final word on Prolog/Epilog

- “Magic” exit codes of 99 and 100 can be used with prolog/epilog scripts
- Exit status code 0 means “success”
- Any other code means failure
  - This can hurt you badly
  - Prolog/Epilog scripts need to be robust and should not throw exit errors lightly
    - Why? Your queues go into E state
SGE Queue Configuration
Queue Configuration

- The usual syntax applies
  - “Show me”
    - `qconf -sq <queue name>`
  - “Let me change it”
    - `qconf -mq <queue name>`
  - “Show me all”
    - `qconf -sql`
# qconf -sq all.q

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gname</td>
<td>all.q</td>
</tr>
<tr>
<td>hostlist</td>
<td>@allhosts</td>
</tr>
<tr>
<td>seq_no</td>
<td>0</td>
</tr>
<tr>
<td>load_thresholds</td>
<td>np_load_avg=4.0</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</td>
</tr>
<tr>
<td>rerun</td>
<td>FALSE</td>
</tr>
<tr>
<td>slots</td>
<td>4, [chrisdag-aliased=4]</td>
</tr>
<tr>
<td>tmpdir</td>
<td>/tmp</td>
</tr>
<tr>
<td>shell</td>
<td>/bin/csh</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>NONE</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>NONE</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>
Some interesting queue params

- **hostlist**
  - Whitespace or comma-separated. Can use hostnames or host groups.
  - *In SGE 6.1 the syntax will get even more flexible*

- **seq_no**
  - Use to influence exec host selection when all other thing are equal

- **load_threshold**
  - When threshold is exceeded, no new jobs are placed on host
  - Can use built-in values or values reported by custom load sensors (example: ‘logged-in-users=5’). Default: “np_load_avg=1.75”

- **suspend_threshold, nsuspend, suspend_interval**
  - Similar to load_threshold but running jobs will actually be suspended/stopped. The ‘nsuspend’ param determines how many jobs per interval get suspend signals. ‘suspend_interval’ defaultsto 00:05:00.
Some interesting queue params

- **qtype**
  - Only “B” or “I” in 6.x
  - Parallel (“P”) and Checkpoint (“C”) are implicit if configured into queue config
- **pe_list**
  - What parallel environment (PE) objects this queue supports
- **slots**
  - Max number of tasks or jobs that this queue supports
- **shell_start_mode**
  - Same behavior as in cluster config
  - `unix_behavior`, `posix_compliant` ...
- **prolog & epilog**
  - Same behavior as in cluster config. Custom scripts that run before|after a job
- **suspend_method, terminate_method, resume_method**
  - Used to override default signals SGE sends
  - Can also configure a path to a script that will run to handle these conditions
Some interesting queue params

- **owner_list**
  - Can delegate queue specific suspend/resume authority to named users

- **user_lists, xuser_lists**
  - Same behavior as in cluster config

- **subordinate_list**
  - Trigger suspension of less important queue instances on same host when value is exceeded
  - Syntax is a bit odd
    - `<queue to suspend> = <slots in THIS queue that must be filled to trigger suspend>`

- **complex_values**
  - Same behavior as in cluster config
Queue Hard & Soft Limits

- **Soft Limits**
  - s_cpu
    - per-process CPU time limit in seconds.
  - s_core
    - per-process maximum core file size in bytes.
  - s_data
    - per-process maximum memory limit in bytes.
  - s_vmem
    - same as s_data (if both are set the minimum is used).

- **Hard Limits**
  - h_cpu
    - per-job CPU time limit in seconds.
  - h_data
    - per-job maximum memory limit in bytes.
  - h_vmem
    - same as h_data (if both are set the minimum is used).
  - h_fsize
    - total number of disk blocks that this job can create.
How soft limits work

1. Job exceeds limit defined by a s_* value
2. Warning signal sent if “notify” is enabled
   - App should trap for these
     - For “s_rt” the signal is SIGXUSR1
     - For “s_cpu” the signal is SIGXCPU
3. If configurable “notify” period passes …
   - Job is sent a SIGSTOP signal (?)
How hard limits work

1. Job exceeds limit defined by a \texttt{h_*} value
2. Warning signal sent if “notify” is enabled
   - App should trap for these
   - When notify is enabled, these are sent before \texttt{SIGKILL}:
     - For “\texttt{h\_rt}” the signal is \texttt{SIGXUSR2}
     - For “\texttt{h\_cpu}” the signal is \texttt{SIGXUSR2}
3. Jobs exceeding \texttt{h_*} get \texttt{SIGKILL} signals
Trivial epilog usage - I

#!/bin/sh
# Simple epilog script

JOB_EXIT_STATUS="\`sed -ne 's/^exit_status=//p' \ $SGE_JOB_SPOOL_DIR/usage | tail -1`"

echo "--------"
echo "Job exited code: $JOB_EXIT_STATUS"
echo "--------"
Trivial epilog usage - II

...  

STARVEDETECT="`grep -c "Licensed number of users already reached" \ $SGE_O_WORKDIR/*.log`"

if [ $STARVEDETECT -gt 0 ]
    then
        echo "License Error Pattern Detected in Output!"
        /bin/tcsh -c "cd $SGE_O_WORKDIR; \ /cl/sw/bin/restart-failed-job.pl"
    else
        echo "No problems detected"
    fi

...
About the exercises

- Normally done live by attendees on demo clusters
  - Helps break up the boredom
- A set of progressively more interesting queue and policy configurations
- Goal: start simple and build towards an ideal configuration
Exercise: Priority Queues #1

- First pass approach
  1. Create 3 queues on your system
     - low.q, regular.q and high.q
  2. Make slot count equal to CPU count
  3. Set load_thresholds to NONE
  4. Set priority values on all queues
     - high.q = -20, low.q = 20
  5. Test all queues with simple.sh
Review - Priority Queues #1

- Queue priority parameter
  - -20 to +20 (*Lower is higher …*)
  - UNIX nice value
  - Has *nothing* to do with scheduling
  - Has *nothing* to do with “qsub -P … “
- Scheduler not looking at load
Review - Priority Queues #1

- What we did
  - Trivial approach to priority queues
  - UNIX nice values applied differently to tasks in each queue will have the effect of “prioritizing” low vs. high vs. regular jobs

- Concerns
  - Leaves “scheduling” to the OS
  - Possible to oversubscribe a system
  - No penalty for misuse of high.q

- We can do better …
Exercise: Priority Queues #2

- Same queue structure as #1
  1. Set notify to 60 for regular.q
  2. Set a soft wall clock limit for regular.q
     - 24 hours (86400 seconds)
  3. Set soft CPU time limit for high.q
     - 9 minutes (540 seconds)
  4. Set a hard CPU time limit for high.q
     - 10 minutes (600 seconds)
Review: Priority Queues #2

Solution
1. qconf -rattr queue notify 60 regular.q
2. qconf -rattr queue s_rt 86400 regular.q
3. qconf -rattr queue s_cpu 540 high.q
4. qconf -rattr queue h_cpu 600 high.q

Discussion
- Main result: user behavior change
  - Unlimited use of low.q, strict limits on high.q
- We can still do better …
Exercise: Priority Queues #3

- Slot hacking
  1. Same queue structure as before
  2. Attach “slots=2” as a host resource on all nodes
  3. Submit test jobs to all queues
Review: Priority Queues #3

- The wizard solution:
  - `qconf -aattr exechost complex_values slots=2 <host>`

- What did we do?
  - Slot limits “solve” the oversubscription problem
  - Still have these problems:
    - FIFO job execution
    - Priority is handled by OS after SGE scheduling

- We can still do better (stay tuned)…
Resources
Resource Hierarchy

Graphic: DanT

SGE training, consulting and special projects - BioTeam Inc. - http://www.bioteam.net
Resources

- Three main types
  - Static
  - Consumable
  - Measured
Common Static Resources

- ‘arch’
- ‘hostname’
- *Custom boolean attribute*
  - nodeLockedLicense=1
Common Consumable Resources

- Free memory
- Available swap space
- Available software license entitlement
Common Measured Resources

- Server load
- Idle time
- Swap usage
- ...

SGE training, consulting and special projects - BioTeam Inc. - http://www.bioteam.net
How users request resources

- Via the “–l” argument
  - Static resource ("arch")
    - qsub -soft -l arch=darwin-x86 ./myJob.sh
  - Custom defined, consumable resource
    - qsub -hard -l ifort=1 ./myCompileScript.sh
Queue associated resources

- qname
- hostname
- notify
- calendar
- min_cpu_interval
- tmpdir
- seq_no

- s_rt
- h_rt
- s_cpu
- h_cpu
- s_data
- h_data
- s_stack
- h_stack
- s_core
- h_core
- s_rss
- h_rss
Host associated resources

- slots
- s_vmem
- h_vmem
- s_fsize
- h_fsize
Partial Complex Listing

```bash
$ qconf -sc
```

<table>
<thead>
<tr>
<th>#name</th>
<th>shortcut</th>
<th>type</th>
<th>relop</th>
<th>requestable</th>
<th>consumable</th>
<th>default</th>
<th>urgency</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td>a</td>
<td>RESTRING</td>
<td>==</td>
<td>YES</td>
<td>NO</td>
<td>NONE</td>
<td>0</td>
</tr>
<tr>
<td>calendar</td>
<td>c</td>
<td>RESTRING</td>
<td>==</td>
<td>YES</td>
<td>NO</td>
<td>NONE</td>
<td>0</td>
</tr>
<tr>
<td>cpu</td>
<td>cpu</td>
<td>DOUBLE</td>
<td>&gt;=</td>
<td>YES</td>
<td>NO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>display_win_gui</td>
<td>dwg</td>
<td>BOOL</td>
<td>==</td>
<td>YES</td>
<td>NO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>h_core</td>
<td>h_core</td>
<td>MEMORY</td>
<td>&lt;=</td>
<td>YES</td>
<td>NO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>h_cpu</td>
<td>h_cpu</td>
<td>TIME</td>
<td>&lt;=</td>
<td>YES</td>
<td>NO</td>
<td>0:0:0</td>
<td>0</td>
</tr>
<tr>
<td>h_data</td>
<td>h_data</td>
<td>MEMORY</td>
<td>&lt;=</td>
<td>YES</td>
<td>NO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>h_fsize</td>
<td>h_fsize</td>
<td>MEMORY</td>
<td>&lt;=</td>
<td>YES</td>
<td>NO</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Anatomy of resource attribute

- **Name**
  - Attribute name

- **Shortcut**
  - Shortcut alias

- **Type**
  - SGE data type
    - Values: INT, DOUBLE, TIME, MEMORY, BOOL, STRING, CSTRING, RESTRING, HOST

- **RelOp**
  - Relational operator
    - Values: “==”, “<”, “>”, “<=”, “>=”
Anatomy of resource attribute

- Requestable
  - Is this something a user can request?
- Consumable
  - Does the resource decrease?
- Default
  - Default value when not explicitly requested
- Urgency
  - Increase entitlement of tasks requesting this attribute via the urgency sub-policy
Anatomy of resource attribute

- **RESTRING**
  - String with regular expression capability
  - 6.0 usage (6.1 expands this a bit …)
    - “*” - Zero or more of any char
    - “?” - Match any one char
    - “.” - This is the “.” char -- no special meaning (!)
    - “\” - Standard escape char
      - "\" = "\"
      - "\*" = "*"
    - “[…]” - Match one of chars within bracket
      - Note: “^” is not interpreted as logical NOT
    - “|” - Logical OR operator
Anatomy of resource attribute

- **RESTRING examples**
  - `-l arch="*x24*|sol*"`
    - **Result:**
      - "arch=lx24-x86" OR "arch=lx24-amd64"
        OR "arch=sol-sparc" OR "arch=sol-sparc64"
        OR "arch=sol-x86" OR ...
  - `-l arch="1x2[4-6]-x86"
    - **Result:**
      - "arch=lx24-x86" OR "arch=lx25-x86" OR "arch=lx26-x86"
Resource Attribute Configuration

- The usual syntax applies
  - “Show me”
    - qconf -sc
  - “Let me change it”
    - qconf -mc
Creating custom attributes

1. Create it in the system complex
2. Associate it with one of the following
   - Queue
     - Add to “complex_values” in queue config
   - Host
     - Add to “complex_values” in host config
   - Global
     - (via special “qconf -me global” host setting)
Cliché Example

- **Scenario:**
  - 5,000 CPU cluster
  - But ...
    - Only 50 commercial licenses for Intel Fortran Compiler ("ifort")
  - The good news
    - Nobody uses ifort outside of the cluster
    - So we don’t need to track usage across the organization
    - Simple limit enforced within SGE will suffice

- **We need to:**
  - Create a user requestable, consumable resource that will limit the use of ifort to no more than 50 concurrent jobs
  - How?
Cliché continued …

1. Add attribute to the SGE complex
   - “qconf -mc”
   - Insert values:
     
     ```
     #name shortcut type relop req cons def urg
     ifort_license ifort INT <= YES YES NONE 0
     ```

2. Associate the attribute to the global host
   - “qconf -me global”
   - Insert value into param:
     
     ```
     complex_values ifort_license=50
     ```
Cliché continued …

- Verify that our attribute is scoped globally
  - “qstat -f -F ifort”

```bash
$ qstat -f -F ifort
queuename                      qtype used/tot. load_avg arch          states
----------------------------------------------------------------------------
all.q@chrisdag-aliased         BIP   0/4       0.99     darwin-ppc            gc:ifort_compiler_license=50
----------------------------------------------------------------------------
testQueue@chrisdag-aliased     BIP   0/4       0.99     darwin-ppc
  gc:ifort_compiler_license=50
```
Cliché continued …

- Test!

$ qsub -cwd -hard -l ifort=2 ./sleeper.sh
Your job 37 ("Sleeper") has been submitted
chrisdag:/tmp dag$
Cliché continued …

- Nothing yet …

```bash
$ qstat -f -F ifort
queuename                qtype used/tot. load_avg arch                      states
-----------------------------------------------
all.q@chrisdag-aliased   BIP  0/4       1.10     darwin-ppc
                          gc:ifort_compiler_lic=50
-----------------------------------------------
testQueue@chrisdag-aliased BIP  0/4       1.10     darwin-ppc
                          gc:ifort_compiler_lic=50
```

# PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS

```
37 0.00000 Sleeper    dag      qw      04/15/2007 17:50:36   1
```
Cliché continued …

- Success!

```bash
$ qstat -f -F ifort
queuename          qtype used/tot. load_avg arch       states
----------------------------------------------------------------------------
all.q@chrisdag-aliased BIP 1/4  1.10 darwin-ppc gc:ifort_compiler_lic=48
                    37  0.55500 Sleeper dag r    04/15/2007 17:50:47 1
----------------------------------------------------------------------------
```
Cliché continued ...

- Test bounds ...

$ qsub -cwd -hard -l ifort=50 ./sleeper.sh
Your job 38 ("Sleeper") has been submitted

$ qsub -cwd -hard -l ifort=50 ./sleeper.sh
Your job 39 ("Sleeper") has been submitted

$ qsub -cwd -hard -l ifort=1 ./sleeper.sh
Your job 40 ("Sleeper") has been submitted
$ qstat -j 39

================================================================================
job_number:                 39
exec_file:                  job_scripts/39
submission_time:            Sun Apr 15 17:54:58 2007
owner:                      dag
uid:                        501
group:                      dag
gid:                        501
sgen:                      /Users/dag
sgen_log_name:              dag
sgen:                      /bin/bash
sgen_workdir:               /private/tmp
sgen_host:                  chrisdag-aliased
account:                    sge
cwd:                        /private/tmp
path_aliases:               /tmp_mnt/ * * /
hard_resource_list:         ifort_compiler_license=50
mail_list:                  dag@chrisdag-aliased
notify:                     FALSE
job_name:                   Sleeper
jobshare:                   0
shell_list:                 /bin/sh
env_list:                   
script_file:                ./sleeper.sh

scheduling info:            (-l ifort_compiler_license=50) cannot run \
globally because it offers only \gc:ifort_compiler_license=0.000000

SGE training, consulting and special projects - BioTeam Inc. - http://www.bioteam.net
Optional Lab Time

- Do this for real on your systems
- Create, test and experiment:
  1. Add attribute to the SGE complex
     - "qconf -mc"
     - Insert values:
       ```
       #name   shortcut  type  relop req cons def  urg
       ifort_license  ifort  INT  <=    YES YES NONE 0
       ```
  2. Associate the attribute to the global host
     - "qconf -me global"
     - Insert value into param:
       ```
       complex_values  ifort_license=50
       ```

SGE training, consulting and special projects - BioTeam Inc. - http://www.bioteam.net
Load Sensors
Load Sensors

- Feed custom data to SGE scheduler
- Can be any executable
- Simple format:
  
  ```
  begin
  host:hostname:value
  end
  ```
Configuring Load Sensors

- Multiple sensors OK
  - Comma separated list
  - Must use full absolute paths
- If “global”
  - Sensor script(s) run on all hosts
- Automatically restarted by SGE
  - If sensor dies
  - If sensor is modified
Sensor scope

- Host sensor can report a host complex
- Global sensor can report a host complex
- Host sensor can report a global complex
- Global sensors should not report global complex values
  - “global” in load sensor speak means “run on every host…”
  - May cause conflict
Custom Load Sensors

- Roll your own …
- Must match particular format
  - Start with value “begin”
  - Each data report on its own line
    - Formatted:
      - `<host|global>:<attributeName>:<value>`
  - End with value “end”
Interactive Login Load Sensor

#!/bin/sh
myhost=`uname -n`

while [ 1 ]; do
    # wait for input
    read input
    result=$?
    if [ $result != 0 ]; then
        exit 1
    fi
    if [ "$input" = "quit" ]; then
        exit 0
    fi
    #send users logged in
    logins=`who | cut -f1 -d " " | sort | uniq | wc -l | sed "s/^ */#/"`
    echo begin
    echo "$myhost:logins:$logins"
    echo end
done

    # we never get here

    exit 0
Interactive Login Load Sensor

$ ./load_sensor.sh

begin
chrisdag-aliased:logins:1
end

begin
chrisdag-aliased:logins:1
end

begin
chrisdag-aliased:logins:1
end

quit
Under documented Sensor Hints

1. The load_sensor configuration parameter will accept multiple comma separated script names

2. An executable program named “qloadsensor” installed into the SGE binary path on any execution host will automatically be run
Exercise

Custom Load Sensor
Exercise: Custom load sensors

1. Create a new complex called “logins”
   *Non-requestable, non-consumable, INT*
2. Get the load_sensor.sh script
3. Configure load_sensor.sh into the global configuration
4. Wait a bit …
5. View the complex status and value(s)
Solution: Custom load sensors

- “qconf -mc”
  - logins al INT <= NO NO 0.0
- “qconf -mconf”
  - Adjust load_sensor param
- “qstat -F logins”
- “qstat -F al”
Discussion: Load sensor exercise

- Scheduling decisions not being made based on “logins:” complex data
- But they could be …
More Queue Config Exercises

Priority Queues #4
Priority Queues #5
Exercise: Priority Queues #4

- Use resources and the Urgency sub-policy
- Create a new resource called “high_priority”
  - Requestable, non-consumable, boolean, Urgency=100
  - Add the new high_priority resource to the high.q configuration
- Do similar for new resource “low_priority”
  - Requestable, non-consumable, boolean, Urgency=-100
  - Add to low.q configuration
- Test
  - Look at urgency information for running/pending jobs
Solution: Priority Queues #4

- Complex entries:
  - high_priority hp BOOL == YES NO FALSE 100
  - low_priority lp BOOL == YES NO FALSE -100

- qconf -aattr queue complex_values hp=TRUE high.q
- qconf -aattr queue complex_values lp=TRUE low.q

- qsub -hard -l hp ...
- qsub -hard -l lp ...

- Watch via “qstat -urg”
Review: Priority Queues #4

- We used the Urgency sub policy
- Jobs inherit the urgency values from the requested resource
  - Multiple resources get summed
- SGE scheduler now handling job prioritization; jobs run in priority order
Review: Priority Queues #4

- Still can do better …
- One annoyance in particular
  - “regular” jobs can still land in low.q or high.q
Forced Requestables

- In the SGE complex …
  - When “requestable=yes”
    - It can be requested by a task/job
  - When “requestable=force”
    - *Must* be requested by a job
    - Huh?
  - Queues or hosts associated with forced requestables become “exclusive”
    - Can only use that host/queue if you request the forced resource
Forced Requestables

- See where we are going here?
- What happens if we designate our “low_priority” and “high_priority” resources as FORCED?
Exercise: Priority Queues #5

1. Edit the high_priority and low_priority resources, set requestable to forced:
   - high_priority hp BOOL == FORCED NO FALSE 100
   - low_priority lp BOOL == FORCED NO FALSE -100

2. Submit test jobs with no queue or resource requests

3. Confirm that these jobs can only land in regular.q

4. See what reason the SGE scheduler gives for not dispatching pending jobs
Review: Priority Queues #5

- Requestable resources
  - *Can* be requested by jobs
- Forced resources
  - *Must* be requested by jobs
- For queues or hosts with forced resources …
  - *Only* jobs requesting that resources can land there
- Effect
  - Jobs can’t run in high.q or low.q without specific user action ("-l hp") or ("-l lp")
  - "qsub -q ... " not required. SGE will figure it out
  - Now we are preventing ‘accidental’ misuse of queues
New topic: Subordinate Queues
New topic: Subordinate Queues

- Subordination is controlled via
  - “subordinate_list” queue attribute
    - Syntax:
      - queue=value (comma separated if multiples)
      - If no value, defaults to slot count for the queue

- When queue has “value” or more jobs active, suspend all the queues in subordinate_list

- When queue has fewer than “value” active jobs, subordinate queues will be resumed

- Potential gotcha
  - Suspended jobs do not relinquish requested resources

*SGE training, consulting and special projects - BioTeam Inc. - [http://www.bioteam.net](http://www.bioteam.net)*
Exercise: Priority Queues #6

- First use of subordinate queues
  1. Delete the host specific “slots” complex from all your nodes if it still exists
  2. Make regular.q and low.q subordinate to high.q
  3. Test

- What happens when high priority jobs are scheduled?
Solution: Priority Queues #6

1. qconf -dattr exechost complex_values slots=4 host
2. qconf -rattr queue subordinate_list regular.q=1 high.q
3. qconf -aattr queue subordinate_list low.q=1 high.q
Priority Queues 1-6 Wrap-up

- Look how far we have come!
- 1st pass: simple UNIX nice hacks
  - FIFO, no protections, oversubscription, etc.
- Final pass:
  - Hard limits on high.q protect abuse
  - Requestable resources w/ urgency entitlements nicely handle prioritization
  - Forced requests keep ‘normal’ jobs in regular.q - users must specifically ask for anything else
- Subordination
- Make sense?
Parallel Environment Configuration
$ 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
Parallel Env Configuration

- **PE issues will also be covered in a later module**
- The usual syntax applies
  - "Show me"
    - `qconf -sp <PE name>`
  - "Let me change it"
    - `qconf -mp <PE name>`
  - "Create new from file"
    - `qconf -Ap ./my-PE-template.txt`
  - "Show me all configured PE’s"
    - `qconf -spl`
Interesting PE parameters

- **start_prog_args & stop_proc_args**
  - Prepare, start and takedown the parallel environment
  - Usually a shell script specific to the parallel implementation; often site specific
  - *Special variables are available in the environment:*
    - `$pe_hostfile`, `$host`, `$job_owner`, `$job_id`, `$job_name`, `$pe`, `$pe_slots`, `$processors`, `$queue`

- **allocation_rule**
  - `<int>` - Fix # of PE slots per host
    - If “1” then all tasks must be on different hosts
  - `$pe_slots` - Force all PE slots to live within the same host
  - `$round_robin` - Rotate through available hosts, maximize host spread
  - `$fill_up` - Minimize host spread by filling up host slots before moving on
Configuring PE environments

- Grid Engine just picks *WHEN* and *WHERE* parallel tasks are placed; admins still must do lots of work.
- Important to understand distinction between
  - Loose PE integration
  - Tight PE integration
- PE implementations are
  - Software specific (MPICH, LAM-MPI, MPICH2), etc.
  - Site specific (highly customized to local environment)
- Online HowTo’s are better than official docs:
  - [http://gridengine.sunssource.net/howto/howto.html](http://gridengine.sunssource.net/howto/howto.html)
User Access
Grid Engine Roles

- **Manager**
  - Can control any aspect of grid engine
  - "qconf -am <user>"

- **Owner**
  - Suspend, resume and disable one or more queues that a user may ‘own’
  - "owner" in Queue conf

- **Operator**
  - Same as manager but no ability to add, delete or change a queue configuration
  - "qconf -ao <user>"

- **User**
  - Can use and query system but can’t change anything
Usersets

- Named sets of users
- Can be used with
  - SGE Roles
  - Access Lists
  - Departments
  - Projects

- Standard syntax
  - qconf -sul
  - qconf -su <userset>
  - ...
  - ...
User Objects

- Needed for policies where individual users must be considered
- Can be created/deleted automatically
  - “enforce_user=auto”
  - Etc.

Standard syntax
- `qconf -suserl`
- `qconf -suser <name>`
- `qconf -duser <name>`
- `qconf -muser <name>`
- ...
Project Objects

- Project affiliation used in several policies
- Allow/deny by userset lists
- A user may be affiliated with more than one project

Standard syntax
- `qconf -sprjl`
- `qconf -sprj <project>`
- `qconf -dprj <project>`
- `qconf -mprj <project>`
- ...

SGE training, consulting and special projects - BioTeam Inc. - http://www.bioteam.net
Department Objects

- Special form of SGE access list
  - Allow functional and override tickets to be applied
- User can only have one department affiliation

- Standard syntax
  - `qconf -sul`
  - `qconf -su <department>`
  - `qconf -du <department>`
  - `qconf -mu <department>`
  - ...

SGE training, consulting and special projects - BioTeam Inc. - http://www.bioteam.net
Path Aliasing

- Allows
  - Resolve inconsistent file paths among different hosts
- Two possible locations:
  - $SGE_ROOT/$SGE_CELL/common/sge_aliases
  - ~/.sge_aliases
- Format (and example)

```bash
#src-path  submit-host  exec-host  dst-path
/Volumes/XRAID/Users  *  *  /Users
```
Default requests (sge_request)

- Allows
  - Per-user or globally defined resource requests
- Three possible locations, sorted by precedence:
  1. $SGE_ROOT/$SGE_CELL/common/sge_request
  2. $HOME/.sge_request
  3. ../sge_request
- "qsub -clear" will wipe any prior active requests
- Trivial per-user example:
  ```
  # always assume current working directory
  -cwd
  # always request Apple/X86 architecture
  -l arch=darwin-x86
  ```
Default requests (sge_qstat)

- Alias in commonly used qstat arguments
- Two possible locations, sorted by precedence:
  1. $SGE_ROOT/$SGE_CELL/common/sge_qstat
  2. $HOME/.sge_qstat
- Command line arguments to qstat trump all else
Exercise - sge_qstat

- Your task:
  - Create a personal .sge_qstat file that
    - Replicates the old SGE 6.0 behavior
      - “qstat” defaults to showing info for all users
Solution - sge_qstat

- $HOME/.sge_qstat
- Contents
  - -u '*'
  - -f -u '*'
Done!