The WTCHG Research Computing Core

Talk 3: Submitting jobs to the cluster

Robert Esnouf (robert@strubi.ox.ac.uk; robert@well.ox.ac.uk),
Jon Diprose (jon@well.ox.ac.uk) & Colin Freeman (cfreeman@well.ox.ac.uk)

Generic emails: support rescomp@well.ox.ac.uk
users rescomp-users@well.ox.ac.uk

A series of introductory talks…

A set of six talks of about 45 minutes each (plus questions)

- Talk 1: What is the ResComp Core?
  (Mon 23/1 10:00 Room B; Robert Esnouf)
- Talk 2: A basic introduction to Linux
  (Wed 25/1 10:00 Room B; Robert Esnouf)
- Talk 3: Submitting jobs to the cluster
  (Thu 26/1 10:00 Room B; Robert Esnouf)
- Talk 4: Monitoring and troubleshooting
  (Mon 30/1 11:00 Room B; Robert Esnouf)
- Talk 5: ResComp centrally-managed applications
  (Wed 1/2 10:00 Room B; Jon Diprose)
- Talk 6: Doing your own thing (compiling and customizing)
  (Thu 2/2 10:00 Room B; Jon Diprose)

The story so far…

Cluster Computing

- A nodes: 288 cores @ 2.67GB/core (32GB/node; 70% C speed)
- B nodes: 640 cores @ 8GB/core (96GB/node; 70% C speed)
- C nodes: 1720 cores @ 16GB/core (256GB/node; 100% C speed)
- D nodes: 768 cores @ 16GB/core (384GB/node; 100% C speed)
- H nodes: 96 cores @ 42.66GB/core (2TB/node; 115% C speed)

Login/submission nodes: rescomp[1-2] and the project servers

- General use: compiling programs, submitting jobs, analysing results

High-performance storage (GPFS)

- /gpfs0: 6.7 TB @ 400MB/s: /users/<*group*>/<*user*>, /apps/well, /mgmt
- /gpfs1 & /gpfs2: 3075 TB @ >20GB/s: /well/<*group*>
- Charging rate £35 / usable TB / 6 months

"Archive" Storage (XFS)

- /arc[1-3][a-g]: 1417TB read-only access. £14.16 / usable TB / 6 months

The story so far…

Basic refresher on key aspects of Linux

- User accounts, groups, UIDs and GIDs
- Commands, shells (bash) and environments
- Environment variables: $PATH and $LD_LIBRARY_PATH
- Standard file streams: stdin, stdout and stderr
- Redirection ("<","<") and pipes ("|")
- Foreground and background jobs ("&"), "nohup" and "screen"

Getting ready for using the cluster

- Writing a shell script
- Executing and sourcing a script
- Keeping track of stdin, stdout and stderr
- Cluster computing is essentially non-graphical and non-interactive!

Overview of this talk…

- Sun Grid Engine (SGE)
- Cluster queues and limits
- An SGE user
- My first cluster job: using qsub
- qsub options
- A first look at qstat
- Priority and the share tree
- SGE monitoring tools

Sun Grid Engine (SGE)

Sun Grid Engine (v. 6.2u5p3) schedules jobs across the cluster

- Free-to-use scheduler (and very simple resource manager)
- Familiar and simple: CLUSTER1, CLUSTER2 & CLUSTER3
- Not developed since Oracle bought Sun (2010) so we are stuck with bugs/features
- Alternatives LSF, PBS, PBSpro, Torque/Maui, slurm...
- "Son of Grid Engine" and Open Grid Engine (Kwiatkowski)
- Univa Grid Engine (UGE) is the commercial fork of SGE about £8-10k/*pa* for a site licence

Sun Grid Engine concepts

Conceptually very simple and resilient to failures

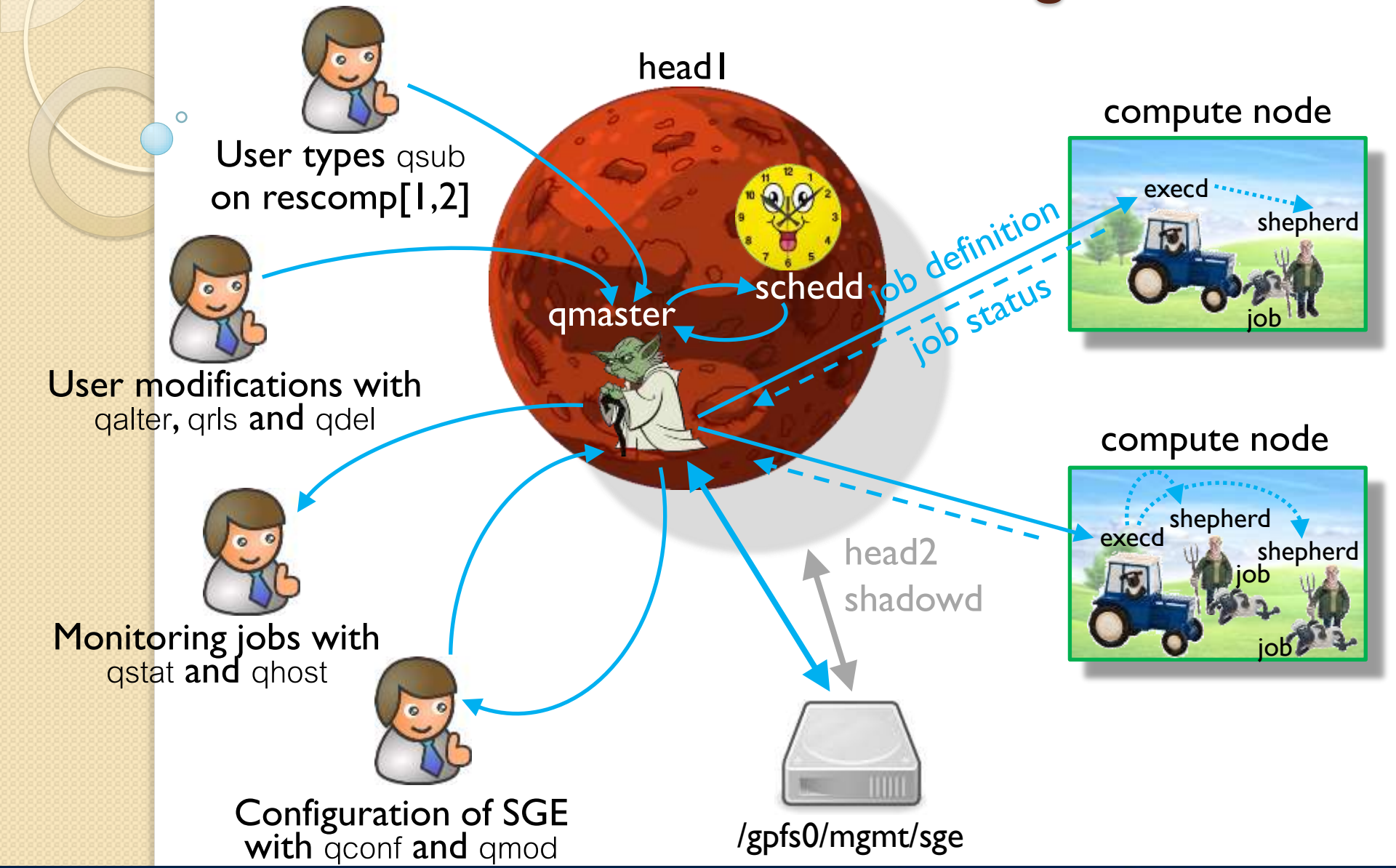
Processes:

- qmaster (on one head node)
- shadowd can start backup qmaster (on other head node)
- qmaster accepts job requests and monitoring requests
- qmaster fires off schedd for scheduling
- qmaster hands highest-priority job to execd on a compute node
- execd creates a job shepherd, detached from execd to run job

No state is held by these processes

- Always written to and/or read from filesystem (/gpfs0/mgmt/sge)
- /gpfs0 is a parallel file system mounted on both head1 and head2
- qmaster, schedd and execd do not need to be running to keep jobs running

Process flow with Sun Grid Engine



Cluster queues and limits

A queue is a definition of a set of rules for running a job

- Name of queue, list of hosts, who has access
- Number of slots (usually 1 per core)
- Run time limits, memory limits *etc.*

Mostly we define two queues per node type

- Short jobs (<24h) and long jobs (<7d)
- 1 core per slot, no use of resource limits ("-l")

Special queues

- For relion jobs, for high memory jobs and on project servers

A queue instance is a queue on a single compute host

- The queue definition defines which queues are on hosts or host groups
- The execd on each host instantiates the queues on that host

General queues on the ResComp cluster

We can (& do) put both short & long queues on a single node

- A resource quota set (rqs) rule limits the total number of queue slots for all queues to the total number of cores on a node

Normal work queues

- short.qa and long.qa: 24h & 7d jobs on "A" nodes, 2.5 GB per slot
- short.qb and long.qb: 24h & 7d jobs on "B" nodes, 8 GB per slot
- short.qc and long.qc: 24h & 7d jobs on "C" & "D" nodes, 16 GB per slot

High-memory queues (access on request)

- himem.qh: 14d jobs on "H" nodes, 42.67 GB per slot

Test queues

- test.qc: 1minute jobs on rescomp[1,2], 16 GB per slot

Project server queues (at the request of the owning groups)

- e.g. coolibah.q: 16 slots, no time limit, on 32-thread machine
- e.g. spencer.q: 40 slots, no time limit, on two 32-thread machines

STRUBI queues on the ResComp cluster

Electron microscopy requires running a job that mixes MPI with threading called relion

- No SGE concept of this type of job, so we need multiple special queues
- When one queue is active on a node, it suspends all the others

STRUBI relion queues

- relion1.qc: 14d MPI jobs on "C" nodes, 16 GB per slot, threading j=1
- relion2.qc: 14d MPI jobs on "C" nodes, 32 GB per slot, threading j=2
- relion4.qc: 14d MPI jobs on "C" nodes, 64 GB per slot, threading j=4
- relion8.qc: 14d MPI jobs on "C" nodes, 128 GB per slot, threading j=8
- relion16.qc: 14d MPI jobs on "C" nodes, 256 GB per slot, threading j=16

STRUBI high-memory node queue

- ginn.qh: 48 slots, no time limit, 5.33GB per slot
- Suspends running jobs on other queues on that node
- himem.qh: 42 slots, 14d jobs on "H" nodes, 42.67 GB per slot

STRUBI project server queue

- belmont.q: 32 slots, no time limit, 8 GB per slot on 32-thread machine

An SGE user

It's not enough merely to have a username on the system, you need to be defined within SGE to get queue access

"Hi, I'm Maurice! What can I do?"

Linux username: schroff

- **Primary group:** "bioinf"
- **Additional groups:** "chimp" **&** "mcveang"

Storage:

- /users/mcvean/schroff **(home directory)**
- /well/bsg, /well/chimp **&** /well/mcvean

SGE username: schroff

- **ACL membership:** mcvean.acl
- **Default project:** mcvean.prjc
- **Additional projects:** mcvean.prja, mcvean.prjb

How to ask SGE about settings with "qconf"

- qconf –suser schroff
- qconf –su mcvean.acl

Running my first cluster job: using qsub

```
[nilufer@login1 ~]$ cat script.sh
#!/bin/bash

echo "*************************************************************"
echo "Run on host: "`hostname`
echo "Operating system: "`uname -s`
echo "Username: "`whoami`
echo "Started at: "`date`
echo "*************************************************************"

R --vanilla << EOD
help()
q()
EOD

echo "*************************************************************"
echo "Finished at: "`date`
echo "*************************************************************"
exit 0
```

If we simply type: qsub script.sh

- Job might run, but how and where?
- Where would the output (or error) go?

All is controlled by "qsub" options

- Entered on the command line, or
- Embedded in the script in lines starting "#$"

Basic qsub options

The shell is controlled by the first line (default is bash)

- #!/bin/bash

Job name should be short and useful

- #$ -N test-job

Project and queue (make sure that they match!)

- #$ -P mcvean.prjc –q short.qc

Destination for stdout and stderr

- #$ -o stdout.log –e stderr.log –j y
- Defaults are $JOB_NAME.[o,e]$JOB_ID(.$SGE_TASK_ID)
- If you specify a directory, files go there with default names

Setting the environment

- #$ -cwd –V
- #$ -pe [shmem|mpi] *n*

Holding a job in the queue

- #$ -h

Running and "array job" of separate "tasks" ($SGE_TASK_ID)

- #$ -t *[m-]n[:s]* –tc *c*

The simple script with qsub options

```
[nilufer@login1 ~]$ cat job.sh
#!/bin/bash

#$ -cwd -V
#$ -N job -j y
#$ -P zondervan.prja -q short.qa
#$ -t 1-10 -tc 2

echo "**********************************************************"
echo "SGE job ID: "$JOB_ID
echo "SGE task ID: "$SGE_TASK_ID
echo "Run on host: "`hostname`
echo "Operating system: "`uname -s`
echo "Username: "`whoami`
echo "Started at: "`date`
echo "**********************************************************"

R --vanilla << EOD
help()
q()
EOD

echo "**********************************************************"
echo "Finished at: "`date`
echo "**********************************************************"
exit 0
```

Running the simple script as a cluster job

With the "qsub" options in the script, it is simple

```
[nilufer@login1 ~]$ qsub job.sh
Your job-array 3202376.1-10:1 ("job") has been submitted
[nilufer@login1 ~]$
```

- SGE gives each job a unique job ID (and each task a unique task ID)

Job enters the scheduler queue and waits for free core(s)

```
   mvdbunt      queued      1    208      1     26  qw=1 (110:110)
   nilufer      queued      1     10      1     10  qw=1 (2700:2700)
  pkalbers      queued     65     65      0      0  qw=65 (14:103)
    tmills      queued     22     22      0      0  qw=22 (26:222)
    ylagou      queued      1  19748      1  19748  qw=1 (2:2)
```

After a while a set of output files are produced

```
[nilufer@login1 ~]$ ls
job.o3202376.1   job.o3202376.2  job.o3202376.4  job.o3202376.6  job.o3202376.8
job.o3202376.10  job.o3202376.3  job.o3202376.5  job.o3202376.7  job.o3202376.9
[nilufer@login1 ~]$
```

Making the SGE job output file useful

Make the output files useful for troubleshooting

- At the "head" of file

```
[nilufer@login1 ~]$ head job.o3202376.2
************************************************************
SGE job ID: 3202376
SGE task ID: 2
Run on host: compA000
Operating system: Linux
Username: nilufer
Started at: Thu Jun 20 09:47:26 BST 2013
************************************************************

R version 2.14.0 (2011-10-31)
```

- At the "tail" of the file

```
[nilufer@login1 ~]$ tail job.o3202376.2

    ## For programmatic use:
    topic <- "family"; pkg_ref <- "stats"
    help((topic), (pkg_ref))


> q()
************************************************************
Finished at: Thu Jun 20 09:47:27 BST 2013
************************************************************
[nilufer@login1 ~]$
```

Simple reasons for a job to go wrong…

The shell script was edited using Windows

- Uses both carriage return & line feed
- User "dos2unix –n file.in file.out"
- Gives *execvp* error in "qstat –j <*job-ID*>"

You have the wrong shell path

- e.g. "#!/usr/bin/csh"
- Also gives *execvp* error in "qstat –j <*job-ID*>"

Environment not set up at submit time

- e.g. in order to use R you need to load the module for the correct version before doing "qsub" with the "–V" and "–cwd" options
- e.g. "module load R/3.1.0"

What happens when a job goes wrong?

- It can disappear from the queue, look with "qacct –j <*job-ID*>"
- It can be re-queued in error state ("Eqw"), look with "qstat –j <*job-ID*>"

How does SGE decide which job to run?

Your job is to submit jobs, our job is to ensure that they start

- Don't wait for free slots, don't expect jobs to start immediately!

We use the "share tree" policy

- Very flexible, but complex, includes historical usage formula
- Another area where there are known SGE bugs
- Essentially "use-it-or-lose-it" (*i.e.* minimal usage history)

Each group has a number of shares for each type of core

- No shares specific to any user

SGE calculates the number of running jobs for each group

For each queued job (in submission order):

- SGE assigns a priority based on how much starting that job would bring the ratios for running jobs for each group toward the ratios of shares
- if equal priorities, then jobs start first come, first served

SGE starts the jobs in priority order, repeating the scheduling calculation regularly

The share tree set-up in more detail

```
strubi.prjb.high=100
strubi.c=20
   strubi.prjc.low=1
      default=1000
   strubi.prjc=10
      default=1000
   strubi.prjc.high=100
tomlinson.a=6
   tomlinson.prja.low=1
      default=1000
   tomlinson.prja=10
      default=1000
   tomlinson.prja.high=100
tomlinson.b=2
   tomlinson.prjb.low=1
      default=1000
   tomlinson.prjb=10
      default=1000
   tomlinson.prjb.high=100
tomlinson.c=30
   tomlinson.prjc.low=1
      default=1000
   tomlinson.prjc=10
      default=1000
   tomlinson.prjc.high=100
yau.c=10
```

Display share tree using "qconf –sst"

Target share at each level of tree

- tomlinson.a=6 (shares on 288 A cores)
- tomlinson.b=2 (shares on 640 B cores)
- tomlinson.c=30 (shares on 1440 C cores)

Priority modification only within group

- tomlinson.prjc.low=1
- tomlinson.prjc=10
- tomlinson.prjc.high=100

Within a group (normal/low priority)

- Take equal turns (round robin)

Within a group (high priority)

- First come, first served

SGE control and monitoring commands

qalter: **change characteristics of a queued/running job**

- **Actually** qalter, qsh, qrsh, qlogin **and** qresub **are all just** qsub

qrls: **release the hold on a job**

qmod: **clear error status on jobs and queues**

qdel [-f]: **delete a job**

qstat: **status of queued and running jobs**

qacct: **accounting records for completed jobs**

qhost: **status of execution hosts (compute nodes)**

qconf: **configuration of SGE**

qalter: changing characteristics of a job

It is possible to change the characteristics of both queued and running jobs

Changes to running jobs may cause them to terminate and/or be re-queued!

Changes to queued jobs are usually either:

- qalter –h u <*jobid*>
- qalter –P group.prja –q short.qa <*jobid*>
- qalter –P group.prja –q short.qa –u <*myname*>

You cannot change the time limit on a running job!

- If it's not going to finish in time then kill it now (qdel)!

qrls: releasing the hold on a job

There are both user and system holds

You can only modify user holds

- Submit jobs with qsub –h or alter them with qalter –h u <*jobid*>
- Release job with qrls <*jobid*>
- Can set job start to depend on the termination of one or more existing jobs with qsub –hold_jid <*joblist*>

qmod: clearing an error state on a job

Occasionally you may be able to fix the error that makes a job go into "Eqw" state

- e.g. disk quota exceeded or if the output directory did not exist
- Clear the job error with qmod -cj <*jobid*>

This is usually a system administrators command

qdel: deleting a job

To delete running or queued jobs use qdel

- qdel <*jobid*>
- qdel –f <*jobid*> **if node is dead or for stuck MPI jobs**

qstat: checking the status of jobs

Report on the status of just my jobs with qstat

- Use qstat –s p for pending jobs and qstat –s r for running jobs

Report on the status of all jobs in the cluster with qstat –u "*"

- Output can be very long and indigestible

```
[nilufer@login1 ~]$ qstat -u ashish
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
3202368 541.66672 R4_baserec ashish       r     06/20/2013 05:55:45 gpfs.qb@compB033.cluster3          2 1
3202368 541.66672 R4_baserec ashish       r     06/20/2013 05:55:45 gpfs.qb@compB033.cluster3          2 3
3202368 541.66672 R4_baserec ashish       r     06/20/2013 05:55:45 gpfs.qb@compB033.cluster3          2 4
3202369 0.00000 R4_printre ashish       hqw   06/20/2013 01:19:55                                  2 1-20:1
3202370 0.00000 R4_reducer ashish       hqw   06/20/2013 01:19:55                                  2 1-20:1
```

Report detailed information on a job with qstat –j <*job_id*>

- Output also long and indigestible

```
script_file:                /gpfs1/well/projects87/got2d/SUMMIT/fund/BamPr/bin4/a.10.base_recalibratorRUN4.sh
parallel environment:  shmem range: 2
jid_predecessor_list (req):  R4_indexdedup20bams
jid_successor_list:          3202369
verify_suitable_queues:     2
project:                    gpfs.prjb
job-array tasks:            1-20:1
usage    1:                 cpu=00:08:36, mem=189.41702 GBs, io=0.39162, vmem=441.582M, maxvmem=441.582M
usage    3:                 cpu=02:33:32, mem=3397.68245 GBs, io=6.01999, vmem=443.020M, maxvmem=443.020M
usage    4:                 cpu=04:35:35, mem=7108.02473 GBs, io=12.96924, vmem=505.566M, maxvmem=505.566M
scheduling info:            queue instance "long.qb@compB017.cluster3" dropped because it is temporarily not available
                            queue instance "short.qb@compB017.cluster3" dropped because it is temporarily not available
                            queue instance "short.qb@compA004.cluster3" is in suspend alarm: load_short=14.280000 (no lo
```

qacct: accounting of finished jobs

When jobs have finished qstat forgets about them

```
[nilufer@login1 ~]$ qstat -j 3202376
Following jobs do not exist:
3202376
[nilufer@login1 ~]$
```

Some records remain accessible through qacct

Aggregated accounting information e.g. for back charging

- Andrew Morris has used just over 25 CPU/years
- We do not back charge for actual use.
- Can take a long time to run!
- Can produce a large amount of output!

```
[nilufer@login1 ~]$ qacct -o amorris
OWNER       WALLCLOCK         UTIME         STIME           CPU             MEMORY                 IO
================================================================================================================
amorris     837830920 713522679.978    707494.717 781348256.501    17751591862.125          252805.412
[nilufer@login1 ~]$
```

You can also get more detailed information on a specific job(s)...

qacct: accounting of a single job or task

Use qacct –j <*job_id*> [-t <*task*>]

- qname: **queue job ran in**
- hostname: **where job ran**
- jobnumber: **job ID**
- taskid: **number of task for array job**

- start_time: **when job started**
- end_time: **when job ended**

- failed: **should be 0**
- exit_status: **exit status of script:**
  - **<128 user defined**
  - **>128 SGE signal**
- 134 = SIGABRT **e.g. memory exceeded**
- 137 = SIGKILL **e.g. time exceeded**

- ru_wallclock: **elapsed time**
- ru_utime: **user cpu time**
- cpu: **cpu time used**
- maxvmem: **maximum size reached by job**

```
[nilufer@login1 ~]$ qacct -j 3202376 -t 2
==============================================================
qname        short.qa
hostname     compA000.cluster3
group        zondervan
owner        nilufer
project      zondervan.prja
department   defaultdepartment
jobname      job
jobnumber    3202376
taskid       2
account      sge
priority     0
qsub_time    Thu Jun 20 09:46:34 2013
start_time   Thu Jun 20 09:47:26 2013
end_time     Thu Jun 20 09:47:27 2013
granted_pe   NONE
slots        1
failed       0
exit_status  0
ru_wallclock 1
ru_utime     0.322
ru_stime     0.058
ru_maxrss    0
ru_ixrss     0
ru_ismrss    0
ru_idrss     0
ru_isrss     0
ru_minflt    13165
ru_majflt    0
ru_nswap     0
ru_inblock   0
ru_oublock   0
ru_msgsnd    0
ru_msgrcv    0
ru_nsignals  0
ru_nvcsw     549
ru_nivcsw    32
cpu          0.380
mem          0.000
io           0.000
iow          0.000
maxvmem      0.000
```

qhost **and** qconf: **housekeeping tools**

qhost: **reports on the state of execution hosts**

- Looking for dead nodes,
- Reporting how many cores are in use
- Reporting how much memory (and swap) are in use
- Checking state of per-node queue instances **e.g.** qhost -q

qconf: **sets and reports on all the administrative setup of SGE**

- qconf –[s|a|A|m|M|d]<*feature*>[l| <*name*>| <*filename*>]

qconf –s<*feature*>l: **list all feature elements**

- qconf –suserl
- qconf -sql

qconf –s<*feature*> <*name*> **reports one element of a feature**

- qconf –suser nilufer
- qconf –sq short.qa
- qconf –shgrp @short.hga

… and now for something a bit simpler

The output of qstat can be seriously verbose!

We have two in-house monitoring tools:

qsum (see qsum –h)

- reformatted from the output of qstat –ext –u "*"
- compactly summarizes running and queueing jobs

qload (see qload –h, qload –ho & qload –hn)

- reformatted from the outputs of qstat –s rs –ext –t –u "*" and qhost –q
- simple text-based diagram of running jobs

Both probably still have bugs in!

Thank you for your attention… any questions?

Next talk 11:00 on Monday… "Monitoring and troubleshooting"