Using the Computing Resources at Gatsby

Dr John Pelan

October 5, 2012

Contents

1 Computational Facilities Overview

The Unit has a range of in-house computational facilities comprised of workstations and central compute servers. Everyone is issued with a reasonably powerful workstation at their desk and these have at least eight x86_64 cores (normally Intel Xeon) and a supplementary GPU device (Nvidia CUDA). Half the resources of each workstation - including the full GPU if installed - are made available as a communal resource accessed by a batch processing system called PBS/Torque.

The central compute facilities are also accessed via the same batch mechanism. You are expected to use the resources available on your workstation first before seeking them elsewhere. You should use the batch queue when your workstation is fully loaded or where the resource requirements are exceeded (e.g. RAM).

If you think you will need additional resources (CPUs, disk-space etc.) please inform the IT Team as soon as you are aware of this. Resources need to be planned for far in advance. and adding memory or disk-space is often not trivial, involves down-time etc. Also, if you are running many jobs in parallel that require sustained disk or network bandwidth you should inform the IT Team. At the very least, you should stagger the jobs or introduce random delays so that they are not all reading and writing at the same time.

1.1 Compute Servers

This is a rough outline of the current facilities (although it may be out-of-date). Normally you do not need to worry about specific machines, just create a batch job that requests the necessary resources and the system will run your job on the most appropriate machine.

Machine Name	CPU cores	Type	Memory (GB)
`deckard`	32	2.1 GHz AMD 6272 (Interlagos)	128
`roy`	32	2.1 GHz AMD 6272 (Interlagos)	128
`rachael`	32	2.1 GHz AMD 6272 (Interlagos)	128
`pris`	32	2.1 GHz AMD 6272 (Interlagos)	128
`zhora`	24	2.2 GHz AMD 6174 (Magny-Cours)	96

Table 1: Compute nodes with 10 GBe interconnect

Machine Name	CPU cores	Type	Memory (GB)	Nvidia CUDA card
`leon`	24	1.8 GHz AMD 6168 (Magny-Cours)	96	Tesla C2050
`behemoth`	48	2.4 GHz AMD 8131 (Istanbul)	256	Quadro FX 3800

Table 2: Compute nodes with CUDA compatible GPUs

1.1.1 Batch Jobs

You can use the PBS batch queue to distribute jobs amongst idle workstations. Please do not use alternative mechanisms (e.g. logging in directly using `ssh`) unless strictly necessary as that will undermine the load-balancing. If you need additional features please submit an enhancement request.

1.1.2 Scratch Space

The `/nfs/data` storage area can be regarded as **temporary scratch space**. Please keep this tidy and delete anything that is no longer needed.

1.1.3 Interconnects

There are no dedicated interconnects for MPI use - however all five machines in Table 1 are interconnected using 10 GbE.

1.2 Compilers & Optimization

If your jobs take a long time, it is worth making some effort to optimize them. This might mean using the Matlab compiler (mex) or coding part/all of your application in Fortran or possibly C. It is also worth investigating the compiler flags. Please note that the Portland Group Compilers are capable of parallelizing C and Fortran programs.

See section 3 for more information on how to compile your code and use these flags.

1.3 Checkpointing

If your jobs run for long periods you should have some checkpointing mechanism (i.e. periodically saving the current state) in case the job is interrupted by urgent system maintenance / failure etc. It is also useful to verify the output of such jobs to ensure that it is in line with expectations.

1.4 Questions

- Why don't you use Condor ? We evaluated a number of systems and PBS/Torque was the easiest to maintain at the time. This is re-evaluated from time to time.
- What about external resources ? We do have access to a number of external resources at UCL and beyond. Please contact the IT Team to establish your requirements.

Unix Path	Windows Path	Description
/	C:	Root partition on a local drive
n/a	My Documents	Do not use!
/unsafe	U:	Short term file storage (days)

Unix Path	Windows Path	Description	Backups?	Highly Available?
/nfs/home*	W:	Home directories, for your most important files	Yes	Yes
/nfs/data*		For data sets or other bulky, reproducable material	No	Yes

2 Filesystems, Quota and Backups

2.1 Considerations

Every computer has a local disk (one that is directly attached) for the purposes of storing the operating system and applications. There are also network based filesystems - these are provided over the network from central fileservers but can be accessed as if they were local.

Files stored on local disks should be considered at **high risk** because they are not protected or backed up in any way. When workstations are replaced or upgraded the local disks are usually erased. In contrast, files stored on the central servers have some level of protection - at the very least they are protected against the failure of a single disk.

It should be clear therefore that **important files should not be stored on local filesystems**. You are free to place **copies** of important files there or use the local files as 'scratch' space but always bear in mind that **local files are ephemeral**. Many Linux distributions are arranged so that the `/tmp` is erased upon every boot.

Some systems are fully redundant, i.e. there is no single point of failure (SPOF) whereas other systems may fail outright. Where there is a SPOF and that fails, it is unlikely that your files will be lost but they will not be available until the system is repaired. This could take many working days, perhaps longer depending on the complexity of the system involved and the support contract.

2.2 Local Filesystems

Do not store important files on the local filesystems.

Windows users should not store files on the desktop.

2.3 Network Filesystems

The following table outlines the network file systems;

As only the `home` directories are regularly backed-up, only these should be used for important files/data. Anything that can be easily recreated or downloaded should be stored in the data filesystems. You are advised to assess your files every 6 months with a view to clearing out old, unnecessary files particularly from the `home` directories.

Common files, for sharing amongst users, can be stored in `/nfs/data[23]` or `/nfs/common`.

2.4 Special Files

Please note that "'core dump files"' (`core.[0-9]*`) may be deleted from time to time so please rename them if you want them retained.

2.5 Disk Usage

A report is generated every day outlining [http://www.gatsby.ucl.ac.uk/local/usage.txt Gatsby disk usage]. You can check your own disk usage using the `du` command under Linux, e.g.

```
cd
du -sk
```

If an NFS filesystem exceeds 95% of its capacity, a message is sent to the top five users of that filesystem asking them to consider reducing their usage.

2.6 System Quota

All filesystems have a finite size and there is a limit on the growth of each one imposed by the tape backup technology. The main purpose of the imposition of quotas is to prevent runaway jobs filling that finite space and thus preventing normal operations. Quota thus has to be set such that the allowed overhead is not so great that only a few users users could fill the filesystem by themselves.

The implementation of quota in Gatsby is such that there is one limit on usage across all the main filesystems (everything prefixed `/nfs/home`. You should actively keep an eye on the amount of space you are consuming as a good citizen.

```
quota -s
```

It will report something like this;

```
Filesystem                      blocks   quota   limit   grace   files   quota   limit   gr
nfs-main:/gatsby/home1/users    12365M  40960M  51200M           90226       0       0
```

Blocks are 512 bytes in size. Your quota is shown in conjunction with a hard limit. You `quota` can be exceeded for brief periods (about 7 days) but the `limit` figure cannot be exceeded. Typically the limit will only be a GB or two over your normal quota but this depends on the overall amount of free space.

Please don't be fooled by;

```
df -H
```

As the disk space is now aggregated it will report a very large number here (and the same number for all central file-systems).

There is a quota report generated daily.

2.6.1 Users' View of Disk Quotas

The quota command provides information on the quotas that have been set by the system administrators and current usage. There are four numbers for each limit: current usage, soft limit (quota), hard limit, and time limit. The soft limit is the number of 1K-blocks (or files) that the user is expected to remain below. The hard limit cannot be exceeded. If a user's usage reaches the hard limit, further requests for space (or attempts to create a file) fail with the "Quota exceeded" (EDQUOT) error.

When a user exceeds the soft limit, the timer is enabled. Any time the quota drops below the soft limits, the timer is disabled. If the timer pops, the particular limit that has been exce eded is treated as if the hard limit has been reached, and no more resources are allocated to the user. The only way to reset this condition, short of turning off limit enforcement or increasing the limit, is to reduce usage below quota. Only the superuser (i.e. a sufficiently capable process) can set the time limits and this is done on a per filesystem basis.

daily.X	Backup from X days ago
weekly.X	Backup from X weeks ago (start of that week)
monthly.X	Backups from X months ago (start of that month)

2.6.2 Surviving When the Quota Limit Is Reached

In most cases, the only way for a user to recover from over-quota conditions is to abort whatever activity is in progress on the filesystem that has reached its limit, remove sufficient fi les to bring the limit back below quota, and retry the failed program.

However, if a user is in the editor and a write fails because of an over quota situation, that is not a suitable course of action. It is most likely that initially attempting to write the file has truncated its previous contents, so if the editor is aborted without correctly writing the file, not only are the recent changes lost, but possibly much, or even all, of the co ntents that previously existed.

There are several possible safe exits for a user caught in this situation. He can use the editor shell escape command to examine his file space and remove surplus files. Alternatively , using `sh`, he can suspend the editor, remove some files, then resume it. A third possibility is to write the file to some other filesystem (perhaps to a file on `/tmp`) where the user's quota has not been exceeded. Then after rectifying the quota situation, the file can be moved back to the filesystem it belongs on.

2.7 Backups

Backups are copies of your files. They are made for two reasons;

1. To guard against accidental modification, corruption or deletion.
2. To guard against disk failure, fire, theft etc.

2.7.1 Online Backup

The first issue above is handled by the online backup system. It is a permanently online, disk backup system which uses rsnapshot. Users can retrieve their own files without any administrator intervention. The files are stored under `/nfs/backup` under directories which indicate when the snapshot was taken;

The timestamps will clearly indicate when the backup was taken. For obvious reasons the backups are **read-only** The number of backups may vary depending on the available disk-space although it is hoped that the last seven (7) days will always be present.

2.7.2 Tape Backup

The second issue is handled by a tape back-up system as the tapes can be physically removed from the Unit. Tape backups are made at least once per month but no more frequently than once per week.

2.7.3 Files Omitted from Backups

The following patterns are used to exclude files from online and tape backups;

```
core.[0-9]*
.nfs*
.netscape/cache/
```

```
matlab\_crash\_dump.*
galeon/Cache/
*.slt/Cache/
.kde/share/thumbnails/
.kde/share/cache/
.kde/share/apps/
.mcop/trader-cache/
.metacity/sessions/
.gconf/%gconf-xml-backend.lock/
.gnome-smproxy-*
gnome-terminal-*
bars\_input/
dir\_knots/
```

3 Compilers & Optimization

3.1 gcc

The GNU compiler is installed on all Linux machines. You should use version 4 of the compiler - check with `gcc -v`.

If code is to be used only on one machine type you should specify this at compile time using `-march=xxxx`. This allows the compiler to make strong assumptions about the processor and its available features to get better results. The option `-mtune=xxxx` tunes for that processor but still allows other CPUs to be used - `march` implies `mtune`.

On the AMD Interlagos machines, for example, one can achieve good results with;

```
gcc -O2 -march=bdver1
```

But the resulting code is only good for those machines.

- AMD Developer Guide to Compiler Optimizations

3.2 PGI

We have two floating licenses for the Portland Group PGI Cluster Development Suite. They are installed under `/opt/pgi` on most machines and are in 32-bit and 64-bit flavours. Please use the modules system to place the PGI compiler binaries on your path.

```
module add pgi
```

The PGI suite includes the AMD Core Math Library (ACML) which are a full implementation of the BLAS and LAPACK libraries, tuned for AMD processors.

- PGI Documentation

3.3 Parallel Matlab & MEX code

Matlab has a Parallel toolbox for programming explicit parallelism.

It is possible to compile your Matlab program as a standalone executable. This may result in a speed-up of your code. It may also be possible to rewrite your Matlab code as Fortran 2008 and compile that instead.

3.4 NAG Libraries

We currently have the NAG library installed - both as a Matlab toolbox and for standalone PGI Fortran applications (`/opt/NAG`).

- NAG libraries

3.5 General Optimization

If you are sweeping parameters but over the same data-set, it should be better to write parallel code (perhaps OpenMP) so that the data-set can be held in common memory and each core sweeps over a subset thereof. This avoids memory pressure and needing to load the same data-set multiple times on the same machine. You will see benefits of caching - depending on memory access patterns.

4 Batch Processing

We use the PBS/Torque & Moab batch queuing system (from Adaptive Computing which distributes jobs across compute nodes and idle workstations. Please do not defeat the queuing system by using alternative load-distribution mechanisms or logging in directly.

Please remember that your jobs may be inadvertently killed if a workstation is switched off or suddenly fails. For jobs that cannot be check-pointed or otherwise need reliability please direct your jobs to the central compute servers using the directives specified below.

Every multi-core workstation should be included by default but the queuing system will only use up to five of those cores (i.e. a load-level of 5.0). There is an upper CPU limit of 10 days of CPU time to catch runaway jobs - this should be more than sufficient.

4.1 Initial Steps

A compute job is just an ordinary shell script which is comprised of the list of commands that will be executed. The preferred shell is `/bin/bash`.

When you submit a job, the job's standard output will go to a file in the current directory. You should get an e-mail from the system if the job fails.

4.2 Essential Tips

- If you are running many jobs in parallel that require sustained disk or network bandwidth you should inform the IT Team. At the very least, you should stagger the jobs or introduce random delays so that they are not all reading and writing at the same time.
- Matlab users need to run in *single-threaded mode* - `matlab -singleCompThread` - to avoid using more CPUs than allocated.
- Similarly, if you are using a multithreaded application you will need to set the number of available CPUs to the number requested. Setting `OMP_NUM_THREADS` to the number of requested CPUs should suffice.
- The queues have maximum limits which cannot be exceeded. Default limits will be applied to all jobs that don't specify what their limits are. You should try to set these appropriately as they will enable the resources to be used efficiently. It will also effect how your jobs are queued.
- The default limits are 1 CPU hr and 1GB of memory. Set your required limits using `qstat` options or via embedded directives in the script.

```
\#PBS -l walltime=1:00:00
\#PBS -l pmem=2gb
```

- NB `pmem` is the memory limit for any one process within the job.
- Mex jobs (compiled standalone Matlab) do not consume licenses.
- Jobs should be launched from NFS directories and not local directories like `/tmp`
- If you are using the modules commands, please include the following preamble:

```
\#PBS -S /bin/bash
#
if [ -f /etc/bash.bashrc ]; then
    . /etc/bash.bashrc
fi
#
```

- Requesting a GPU

```
#PBS -l gres=gpu:1
```

4.3 Commands

- Submit: `qsub jobname`
- Check status of your jobs: `qstat`
- Overview of systems as a whole: `qstat -Q`

More help can be obtained on the command-line from `command --help` or `man command` or via the online manuals from Adaptive Computing.

4.4 Example Scripts

This is a test script.

```
#PBS -S /bin/sh
#
echo "Job execution host: " $HOSTNAME
echo "Shell: " $SHELL
echo "The host where the job was submitted: " $PBS\_O\_HOST
echo "The login name was used for submitting the job: " $PBS\_O\_LOGNAME
echo "Home directory: " $PBS\_O\_HOME
echo "Working directory: " $PBS\_O\_WORKDIR
echo "PBS environment: " $PBS\_ENVIRONMENT
echo "Original queue: " $PBS\_O\_QUEUE
echo "Execution queue: " $PBS\_QUEUE
echo "Job ID: " $PBS\_JOBID
echo "Job name: " $PBS\_JOBNAME
\#
```

A generic Matlab script

```
#PBS -S /bin/bash
#PBS -l pmem=1gb
#
echo "Job execution host: " $HOSTNAME
#
```

```
if [ -f /etc/bash.bashrc ]; then
    . /etc/bash.bashrc
fi
#
#
module add matlab
matlab -nodisplay -singleCompThread -r "a = sum(rand([1,10000])); save a; exit;"
#
```

4.5 Example Parameter Sweeps

There are many ways of doing this and it will depend on the nature of your job. The easiest way is to use the built-in `-t array_request` function in qsub:

```
The  array_request argument is an integer id or a range of integers. Multiple ids or id
ranges can be combined in a comma delimted list. Examples : -t 1-100 or -t 1,10,50-100
```

Each job will have `PBS_ARRAYID` set to a unique value according to what was specified above:

```
#PBS -S /bin/bash
#PBS -l pmem=5gb
#
echo $PBS_ARRAYID
#
# You can read the environment variable in Matlab using getenv('PBS_ARRAYID')
#
```

4.5.1 Parameter Sweep - Alternative

You can create a unique script for each job.

```
#!/bin/tcsh -f
#
foreach parameter ( 1 2 3 4 9 11 27 45 87 99 )
#
# A separate M file is created per parameter
#
cat > in.${parameter}.m   << EOF
subject=$parameter
run_some_function_of_subject
EOF
#
# A separate script (qsub job) is created per parameter
#
cat > script.${parameter} << EOF
#!/bin/tcsh -f
#
```

```
/opt/matlab-7.7.0/bin/matlab -singleCompThread -nodisplay -nosplash >  out.${subject}  < in
EOF
#
# Make the script executable (useful for testing)
#
chmod a+x script.${subject}
#
# Submit the job
#
qsub script.${subject}
#
end
```

4.6 Reservations

You can reserve your own workstation by submitting a dummy job.

```
#PBS -l walltime=1:00:00
#PBS -l mem=2gb
#PBS -l pmem=2gb
#PBS -l nodes\section{1:ppn}4
#PBS -l host=flies
#
# Reserve 4 CPUS and 2GB on flies for 1 hour
# Just edit this file to suit your needs
#
sleep 3600
```

You can use this script to generate the reservation jobs (courtesy of Charles & John) ;

```
#!/bin/sh
# usage:
#   reserve [<memory>  [<cores> [<hours>]]]
#       memory defaults to 2gb
#       cores defaults to 4
#       hours defaults to 1
#  this reserves 2gb of memory, 4 cores for one hour on the current
#  machine.
mem=${1:-2gb}
cores=${2:-4}
hours=${3:-1}
seconds=$((hours*60*60))
trap 'rm $tmp' INT QUIT TERM EXIT
tmp=`mktemp`
host=`hostname`
cat <<EOF >$tmp
#PBS -l walltime\section{$hours:00:00,mem}$mem,pmem\section{$pmem,nodes}1:ppn\section{$cores,host}$ho
```

```
sleep $seconds
EOF
qsub $tmp
```