Using the Yale HPC Clusters

Robert Bjornson

Yale Center for Research Computing
Yale University

Oct 2016

What is the Yale Center for Research Computing?

- Independent center under the Provost's office
- Created to support your research computing needs
- Focus is on high performance computing and storage
- ~15 staff, including applications specialists and system engineers
- Available to consult with and educate users
- Manage compute clusters and support users
- Located at 160 St. Ronan st, at the corner of Edwards and St. Rona
- http://research.computing.yale.edu

Y|CRC

What is a cluster?

A cluster usually consists of a hundred to a thousand rack mounted computers, called **nodes**. It has one or two login nodes that are externally accessible, but most of the nodes are compute nodes and are only accessed from a login node via a batch queueing system (BQS), also called a **job scheduler**.

The CPU used in clusters may be similar to the CPU in your desktop computer, but in other respects they are rather different.

- Linux operating system
- Command line oriented
- Many cores (cpus) per node
- No monitors, no CD/DVD drives, no audio or video cards
- Very large distributed file system(s)
- Connected internally by a fast network

Why use a cluster?

Clusters are very powerful and useful, but it may take some time to get used to them. Here are some reasons to go to that effort:

- Don't want to tie up your own machine for many hours or days
- Have many long running jobs to run
- Want to run in parallel to get results quicker
- Need more disk space
- Need more memory
- Want to use software installed on the cluster
- Want to access data stored on the cluster

Limitations of clusters

Clusters are not the answer to all large scale computing problems. Some of the limitations of clusters are:

- Cannot run Windows programs
- Not for persistent services (DBs or web servers)
- Not really intended for interactive jobs (especially graphical)
- Jobs that run for weeks can be a problem (unless checkpointed)

Summary of Yale Clusters

	Omega	Grace	Louise/Farnam	Ruddle
Role	FAS	FAS	LS/Med	YCGA
Total nodes	1028	216	115+	156+
Cores/node	8	20	mostly 20	
Mem/node	36 GB/48 GB	128 GB	128-1500GB	
Network	QDR IB	FDR IB	10 Gb EN	
File system	Lustre	GPFS	GPFS	NAS + GPFS
Batch queueing	Torque	LSF	Slurm	Torque/Slurm

Details on each cluster here:
`http://research.computing.yale.edu/hpc-clusters`

Setting up a account

Accounts are free of charge to Yale researchers.

Request an account at:
`http://research.computing.yale.edu/account-request`.

After your account has been approved and created, you will receive an email describing how to access your account. This may involve setting up ssh keys or using a secure login application. Details vary by cluster.

If you need help setting up or using ssh, send an email to: `hpc@yale.edu`.

Ssh to a login node

To access any of the clusters, you must use **ssh**. From a Mac or Linux machine, you simply use the **ssh** command:

```
laptop$ ssh netid@omega.hpc.yale.edu
laptop$ ssh netid@grace.hpc.yale.edu
laptop$ ssh netid@louise.hpc.yale.edu
laptop$ ssh netid@farnam.hpc.yale.edu
laptop$ ssh netid@ruddle.hpc.yale.edu
```

From a Windows machine, you can choose from programs such as PuTTY or X-Win32, both of which are available from the Yale Software Library: `http://software.yale.edu`..

For more information on using PuTTY (and WinSCP), go to: `http://research.computing.yale.edu/faq#ssh`

Y|CRC

Ssh key pairs

- We use key pairs instead of passwords to log into clusters
- Keys are generated as pair: public (id_rsa.pub) and private (id_rsa)
- You should always use a pass phrase to protect private key!
- You can freely give the public key to anyone
- NEVER give the private key to anyone!

Sshing to Farnam and Ruddle

- Farnam and Ruddle have an additional level of ssh security, using Multi Factor Authentication (MFA)
- We use Duo, the same MFA as other secure Yale sites

Example:

```
bjornson@debian:~$ ssh rdb9@ruddle.hpc.yale.edu
Enter passphrase for key '/home/bjornson/.ssh/id_rsa':
Duo two-factor login for rdb9

Enter a passcode or select one of the following options:

 1. Duo Push to XXX-XXX-9022
 2. Phone call to XXX-XXX-9022
 3. SMS passcodes to XXX-XXX-9022

Passcode or option (1-3): 1
Success. Logging you in...
```

YCRC

Running jobs on a cluster

Two ways to run jobs on a cluster:
Interactive:

- you request an allocation
- system grants you one or more nodes
- you are logged onto one of those nodes
- you run commands
- you exit and system automatically releases nodes

Batch:

- you write a job script containing commands
- you request an allocation
- system grants you one or more nodes
- your script is automatically run on one of the nodes
- your script terminates and system releases nodes
- system sends a notification via email

Y|CRC

Interactive vs. Batch

Interactive jobs:

- like a remote session
- require an active connection
- for development, debugging, or interactive environments like R and Matlab

Batch jobs:

- non-interactive
- can run many jobs simultaneously
- your best choice for production computing

Interactive allocations

```
Torque (omega) : qsub -I -q fas_devel
Torque (louise): qsub -I -q general
Torque (ruddle): qsub -I -q interactive
LSF    (grace) : bsub -Is -q interactive bash
Slurm  (farnam): srun -p interactive --pty bash
```

You'll be logged into a compute node and can run your commands. To exit, type exit or ctrl-d

Torque: Example of an interactive job

```
laptop$ ssh sw464@omega.hpc.yale.edu
Last login: Thu Nov  6 10:48:46 2014 from akw105.cs.yale.internal
[ snipped ascii art, etc ]
login-0-0$ qsub -I -q fas_devel
qsub: waiting for job 4347393.rocks.omega.hpc.yale.internal to start
qsub: job 4347393.rocks.omega.hpc.yale.internal ready
compute-34-15$ cd ~/workdir
compute-34-15$ module load Apps/R/3.0.3
compute-34-15$ R --slave -f compute.R
compute-34-15$ exit
login-0-0$
```

LSF: Example of an interactive job

```
laptop$ ssh sw464@grace.hpc.yale.edu
Last login: Thu Nov  6 10:47:52 2014 from akw105.cs.yale.internal
[ snipped ascii art, etc ]
grace1$ bsub -Is -q interactive bash
Job <358314> is submitted to queue.
<<Waiting for dispatch ...>>
<<Starting on c01n01>>
c01n01$ cd ~/workdir
c01n01$ module load Apps/R/3.0.3
c01n01$ R  --slave -f compute.R
c01n01$ exit
login-0-0$
```

Slurm: Example of an interactive job

```
farnam-0:~ $ srun -p general --pty bash
c01n01$ module load Apps/R
c01n01$ R --slave -f compute.R
c01n01$ exit
farnam-0:~ $
```

Example batch script (Torque)

Here is a very simple batch script that executes an R script. It can contain PBS directives that embed qsub options in the script itself, making it easier to submit. These options can be overridden by the command line arguments, however.

```
#!/bin/bash
#PBS -q fas_normal
#PBS -m abe -M stephen.weston@yale.edu
cd $PBS_O_WORKDIR

module load Apps/R/3.0.3
R --slave -f compute.R
```

Y|CRC

Example of a non-interactive/batch job (Torque)

```
login-0-0$ qsub batch.sh
4348736.rocks.omega.hpc.yale.internal
-bash-4.1$ qstat -1 -n 5262300

rocks.omega.hpc.yale.internal:
                                                                  Req'd    Req'd       Elap
Job ID     Username    Queue    Jobname     SessID  NDS   TSK    Memory   Time    S   Time
-------    ----------- -------- ----------- ------ ----- ------ ------ --------- - ---------
5262300    sw464       fas_norm batch.sh    19620   --     --     --   01:00:00 R  00:00:34   compute-34-15/2
```

Example batch script (LSF)

The same script in LSF:

```
#!/bin/bash
#BSUB -q shared

# automatically in submission dir
module load Apps/R/3.0.3
R --slave -f compute.R
```

Example of a batch job (LSF)

```
[rdb9@grace0 ~]$ bsub < batch.sh
Job <113409> is submitted to queue <shared>.
[rdb9@grace0 ~]$ bjobs 113409
JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
113409     rdb9    RUN   shared     grace0      c13n12      *name;date Sep  7 11:18
```

Note that "batch.sh" was specified via input redirection. This is required when using BSUB directives in the script, otherwise the directives will be ignored.

Slurm: Example of a batch script

The same script in Slurm:

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -c 20 -N 1
#SBATCH -t 00:30:00
#SBATCH -J testjob

module load Apps/R
R --slave -f compute.R
```

Slurm: Example of a batch job

```
$ sbatch test.sh
Submitted batch job 42
$ squeue -j 42
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
   42   general   my_job     rdb9  R       0:03      1 c13n10
```

The script runs in the current directory. Output goes to slurm-*jobid*.out by default, or use -o

Copy scripts and data to the cluster

- On Linux and Mac, use scp and rsync to copy scripts and data files from between local computer and cluster.

```
laptop$ scp -r ~/workdir netid@omega.hpc.yale.edu:
laptop$ rsync -av ~/workdir netid@grace.hpc.yale.edu:
```

- For Mac, Cyberduck is a good graphical tool. `https://cyberduck.io`
- For Windows, WinSCP is available from the Yale Software Library: `http://software.yale.edu`.
- Other windows options include:
  - Bitvise ssh `https://www.bitvise.com`
  - Cygwin (Linux emulator in Windows: `https://www.cygwin.com`

Summary of Torque commands (Omega/Louise/Ruddle)

Description	**Command**
Submit a job	qsub [*opts*] *SCRIPT*
Status of a job	qstat *JOBID*
Status of a user's jobs	qstat -u *NETID*
Detailed status of a user's jobs	qstat -1 -n -u *NETID*
Cancel a job	qdel *JOBID*
Get queue info	qstat -Q
Get node info	pbsnodes *NODE*

Summary of qsub options

Description	**Option**
Queue	-q *QUEUE*
Process count	-l nodes=*N*:ppn=*M*
Wall clock limit	-l walltime=*DD*:*HH*:*MM*:*SS*
Memory limit	-l mem=*J*gb
Interactive job	-I
Interactive/X11 job	-I -X
Job name	-N *NAME*

Examples:

```
login-0-0$ qsub -I -q fas_normal -l mem=32gb,walltime=24:00:00
login-0-0$ qsub -q fas_long -l mem=32gb,walltime=3:00:00:00 job.sh
```

Summary of LSF commands (Grace)

Description	**Command**
Submit a job	bsub [*opts*] <*SCRIPT*
Status of a job	bjobs *JOBID*
Status of a user's jobs	bjobs -u *NETID*
Cancel a job	bkill *JOBID*
Get queue info	bqueues
Get node info	bhosts

Summary of LSF bsub options (Grace)

Description	**Option**
Queue	-q *QUEUE*
Process count	-n *P* -R "span[hosts=1]"
Process count	-n *P*
Wall clock limit	-W *HH*:*MM*
Memory limit	-M *M*
Interactive job	-Is bash
Interactive/X11 job	-Is -XF bash
Job name	-J *NAME*

Examples:

```
grace1$ bsub -Is -q shared -M 32768 -W 24:00     # 32gb for 1 day
grace1$ bsub -q long -M 32768 -W 72:00 < job.sh  # 32gb for 3 days
```

Summary of Slurm commands (Farnam)

Description	**Command**
Submit a batch job	sbatch [*opts*] *SCRIPT*
Submit an interactive job	srun -p interactive --pty [*opts*] bash
Status of a job	squeue -j *JOBID*
Status of a user's jobs	squeue -u *NETID*
Cancel a job	scancel *JOBID*
Get queue info	squeue -p *PART*
Get node info	sinfo -N

Summary of Slurm sbatch/srun options (Farnam)

Description	**Option**
Partition	-p *QUEUE*
Process count	-c *Cpus*
Node count	-N *Nodes*
Wall clock limit	-t *HH*:*MM* or *DD*-*HH*
Memory limit	–mem-per-cpu *M* (MB)
Interactive job	srun --pty bash
Job name	-J *NAME*

Examples:

```
farnam1$ srun --pty -p general --mem-per-cpu 32768 -t 24:00 bash # 32gb for 1 day
farnam1$ sbatch -p general --mem-per-cpu 32768 -t 72:00 job.sh   # 32gb for 3 days
```

Controlling memory usage

- It's crucial to understand memory required by your program.
- Nodes (and thus memory) are often shared
- Jobs have default memory limits that you can override
- Each BQS specifies this differently

To specify 16 cores with 20 GB each:

```
Torque: qsub -lnodes=2:ppn=8 -lmem=180gb t.sh
LSF: bsub -n 16 -M 20000 t.sh
Slurm: sbatch -n 16 --mem-per-cpu=20000 t.sh
```

Y|CRC

Controlling walltime

- Each job has a default maximum walltime
- The job is killed if that is exceeded
- You can specify longer walltime to prevent being killed
- You can specify shorter walltime to get resources faster

To specify walltime limit of 2 days:

```
Torque: qsub -lwalltime=2:00:00:00 t.sh
LSF: bsub -W 48:00 t.sh
Slurm: sbatch -t 2- t.sh
```

Y|CRC

Submission Queues

Regardless of which cluster you are on, you will submit jobs to a particular queue, depending on:

- Job's requirements
- Access rights for particular queues
- Queue rules

Each cluster has its own queues, with specific rules, such as maximum walltime, cores, nodes, jobs, etc. We detail them in the following slides.

Omega Queues

You must always specify a queue via the qsub **-q** argument on Omega. The primary factor in choosing a queue is the value of **walltime** because the different queues have different restrictions on how long jobs can run.

fas_very_long	4 weeks
fas_long	3 days
fas_high	1 day
fas_normal	1 day
fas_devel	4 hours

Grace Queues

The default queue on Grace is **shared**, but you can specify a different queue via the bsub -**q** argument. The primary factor in choosing a queue is the value of the -**W** option because the different queues have different restrictions on how long jobs can run.

long	4 weeks
week	1 week
shared	1 day
interactive	4 hours

Y|CRC

Louise Queues

- Louise has 5 different nodes types, ranging from 4-64 cores, and 16-512 GB of RAM. General queue jobs may be run on any of these node types. Use -W PARTITION:*type* to select a specific type.

Queue	**Walltime def/max (days)**	**MAX jobs/cpus**	**Notes**
general	30/30	10/64	default
scavenge	30/30	4/320	preemptable
PI	none	none	members only

Y|CRC

Farnam Queues

Queue	Walltime def/max	MAX cpus	Notes
general	7/30	100	default
interactive	1/1	4	
bigmem	1/7	64	1.5TB RAM
gpu	1/7	TBD	2 GPUS/node
scavenge	1/7	200	preemptable
PI	14/14	N/A	members only

Ruddle Queues

Queue	Walltime def/max	MAX cpus	Notes
default	7/30	300	default
interactive	1/1	20	
bigmem	7/30	64	1.5TB RAM

Module files

Much of the HPC software on Omega, Grace, Ruddle, and Farnam is installed in non-standard locations. This makes it easier to to maintain different versions of the same software allowing users to specify the version of an application that they wish to use. This is done with the module command.

For example, before you can execute Matlab, you need to initialize your environment by loading a Matlab "module file":

```
$ module load Apps/Matlab
```

This will modify variables in your environment such as PATH and LD_LIBRARY_PATH so that you can execute the **matlab** command.

Omega: Finding module files

There are many module files for the various applications, tools, and libraries installed on the FAS clusters. To find the module file that will allow you to run Matlab, use the ‘modulefind‘ command:

```
login-0-0$ modulefind matlab
```

This will produce output like:

```
/home/apps/fas/Modules:
Apps/Matlab/R2010b
Apps/Matlab/R2012b
Apps/Matlab/R2014a
```

You can get a listing of available module files with:

```
login-0-0$ module avail
```

Grace, Ruddle, Farnam: Finding module files

There are many module files for the various applications, tools, and libraries installed on thw clusters. To find the module file that will allow you to run Bowtie, use the 'module avail' command:

```
[rdb9@grace0 ~]$ module avail matlab

---------------------------------------------------------- /gp
   Apps/Matlab/R2013b    Apps/Matlab/R2014a    Apps/Matlab/R2015a (I

  Where:
   (D):  Default Module
```

You can get a listing of all available module files with:

```
grace1$ module avail
```

Example "module load" commands

Here are some "module load" commands for scripting languages:

```
module load Langs/Python
module load Langs/Perl
module load Apps/R
module load Apps/Matlab
module load Apps/Mathematica
```

You can specify a specific version:

```
module load Langs/Python/2.7.6
```

Run your program/script

When you're finally ready to run your script, you may have some trouble determining the correct command line, especially if you want to pass arguments to the script. Here are some examples:

```
compute-20-1$ python compute.py input.dat
compute-20-1$ R --slave -f compute.R --args input.dat
compute-20-1$ matlab -nodisplay -nosplash -nojvm < compute.m
compute-20-1$ math -script compute.m
compute-20-1$ MathematicaScript -script compute.m input.dat
```

You often can get help from the command itself using:

```
compute-20-1$ matlab -help
compute-20-1$ python -h
compute-20-1$ R --help
```

Y|CRC

Running graphical programs on compute nodes

Two different ways:

- X11 forwarding
  - easy setup
  - ssh -Y to cluster, then qsub -Y/bsub -XF/srun –x11
  - works fine for most applications
  - bogs down for very rich graphics
- Remote desktop (VNC)
  - more setup
  - allocate node, start VNC server there, connect via ssh tunnels
  - works very well for rich graphics

Y|CRC

Cluster Filesystems

Each cluster has a number of different filesystems for you to use, with different rules and performance characteristics.

It is very important to understand the differences. Generally, each cluster will have:

Home	:	Backed up, small quota, for scripts, programs, documents, etc.
Scratch	:	Not backed up. Automatically purged. For temporary files.
Project	:	Not backed up. For longer term storage.
Local HD	:	/tmp For local scratch files.
RAMDISK	:	/dev/shm For local scratch files.
Storage@Yale	:	University-wide storage (active and archive).

Consider using local HD or ramdisk for intermediate files. Also consider avoiding files by using pipes.
For more info:
`http://research.computing.yale.edu/hpc/faq/io-tutorial`

Y|CRC

Use Links to avoid copying fastq files

Many users incur a 2-3x space increase:

```
$ zcat Sample*_R2_0??.fastq.gz >  /scratchspace/Sample.fastq
```

Use soft links instead

```
$ mkdir Project/Sample_7777
$ cd Project/Sample_7777
$ ln -s /path/to/sample/*.fastq.gz .
```

Many programs can use compressed split files directly. If they can't, use this trick

```
$ bwa ... <(zcat *R1_*.fastq.gz) <(zcat *R2_*.fastq.gz)
```

Use Pipes to improve performance

Faster: avoids file IO and increases parallelism

Using Files

```
$ gunzip test_export.txt.gz
$ perl filter.pl test_export.txt \
  test_filtered.txt
$ perl illumina_export2sam.pl \
  --read1=test_filtered.txt > test_filtered.sam
$ samtools view -bS -t hg19.fa.fai \
  test_filtered.sam -o test_filtered.bam
$ samtools sort test_filtered.bam test_sorted
```

Using Pipes

```
$ gunzip -c test.export.txt.gz \
| perl filter.pl - - \
| perl illumina\_export2sam.pl --read1=- \
| samtools view -bS -t hg19.fa.fai - \
| samtools sort - test.sorted
```

Y|CRC

Genome references installed on Ruddle

Please do not install your own copies of popular files (e.g. genome refs).

We have a number of references installed here:

`/home/bioinfo/genomes`

If you don't find what you need, please ask us, and we will install them.

Wait, where is the Parallelism?

Qsub or Bsub can allocate multiple nodes, but the script runs on one core on one node sequentially. How do we use multiple cpus?

- Submit many batch jobs simultaneously (not good)
- Use job arrays (better)
- Submit a parallel version of your program (great if you have one)
- Use SimpleQueue (excellent)

SimpleQueue

- Useful when you have many similar, independent jobs to run
- Automatically schedules jobs onto a single PBS allocation

Advantages

- Handles startup, shutdown, errors
- Only one batch job to keep track of
- Keeps track of status of individual jobs
- Automatically schedules jobs onto a single PBS allocation

Y|CRC

Job Arrays

- Useful when you have many nearly identical, independent jobs to run
- Starts many copies of your script, distinquished by a task id.

Submit jobs like this:

```
Torque: qsub -t 1-100 ...
Slurm: sbatch --array=1-100 ..
LSF: bsub -J "job [1-100]" ...
```

Script references an environment variable:

```
#PBS -t 1-10

cd $PBS_O_WORKDIR

./mycommand -i input.${PBS_ARRAYID} \
    -o output.${PBS_ARRAYID}
```

Other BQS are similar

Using SimpleQueue

1. Create file containing list of commands to run (jobs.list)

```
cd ~/mydir/myrun; prog arg1 arg2 -o job1.out
cd ~/mydir/myrun; prog arg1 arg2 -o job2.out
...
```

2. Create launch script

```
# Louise:
/path/to/sqPBS.py queue Nodes Account Title jobs.list > run.sh
# Omega, Grace, Ruddle, Farnam:
module load Tools/SimpleQueue
sqCreateScript -q queuename -n 4 jobs.list > run.sh
```

3. Submit launch script

```
qsub run.sh     # Louise, Omega, Ruddle
bsub < run.sh   # Grace
sbatch run.sh   # Farnam
```

For more info, see
http://research.computing.yale.edu/hpc/faq/simplequeue

Y|CRC

Best Practices

- Start Slowly
  - Run your program on a small dataset interactively
  - In another ssh, watch program with top. Track memory usage.
  - Check outputs
  - Only then, scale up dataset, convert to batch run
- Input/Output
  - Think about input and output files, number and size
  - Should you use local or ram filesystem?
- Memory
  - Use top or /usr/bin/time -a to monitor usage
  - Consider using memory option when submitting or allocate entire node.
- Be considerate! Clusters are shared resources.
  - Don't run programs on the login nodes. Make sure to allocate a compute node.
  - Don't submit a large number of jobs at once. Use simplequeue.
  - Don't do heavy IO to /home.
  - Don't fill filesystems.

Y|CRC

Plug for scripting languages

- Learning basics of a scripting language is a great investment.
- Very useful for automating lots of day to day activities
  - Parsing data files
  - Converting file formats
  - Verifying collections of files
  - Creating processing pipelines
  - Summarizing, etc. etc.
- Python (strongly recommended)
- Bash
- Perl (if your lab uses it)
- R (if you do a lot of statistics or graphing)

Y|CRC

To get help

- Send an email to: `hpc@yale.edu`
- Read documentation at:
  `http://research.computing.yale.edu/hpc-support`
- Email us directly:
  - Stephen.weston@yale.edu, Office hours at CSSSI on Wednesday morning from 9 to 12 or by appointment
  - Robert.bjornson@yale.edu, By appointment

Y|CRC

Resources

- Please feel free to contact us.
- Online documentation: `http://research.computing.yale.edu/hpc`
- Very useful table of equivalent BQS commands: https://slurm.schedmd.com/rosetta.pdf
- Recommended Books
  - Learning Python: Mark Lutz
  - Python Cookbook: Alex Martelli
  - Bioinformatics Programming using Python: Mitchell Model
  - Learning Perl: Randal Schwarz

Y|CRC