Distributed Computing with ICC

James Balamuta

Department of Informatics, Statistics
University of Illinois at Urbana-Champaign

July 31, 2017

On the Agenda

Available Clusters

- **Illinois Campus Cluster (ICC)**: Follows a time share model with a majority of departments buying in and can be used for classes. *Try this option first*.
- **Keeling** (formerly **manabe**): LAS machine for faculty + grads used as a stepping stone to ICC
- **ROGER**: Launched about 2 years ago specializing in geospatial and Hadoop oriented jobs
- **Biocluster**: Open to a majority of departments with preference to biology fields, bills for each job run.
- **BlueWaters**: Expensive, but grants can be had if faculty are affiliated with NCSA. Requires two-factor authentication

What is ICC?

- Illinois Campus Cluster (ICC), there are two unique systems: Taub and Golub. The latter is a newer deployment (2013 vs. 2011).
- Taub is being removed as it is now obsolete.
- The cluster has **530+ computing nodes available for use**.
- These nodes are managed by Torque Resource Manager, a form of OpenPBS, with the Moab Workload Manager.
- Management of nodes relate to two forms of queues for job submission:
  - **Primary:** Settings specific to the investor.
  - **Secondary:** Shared resource queue that allows access to any idle nodes in the cluster under specific limits (see next slide).

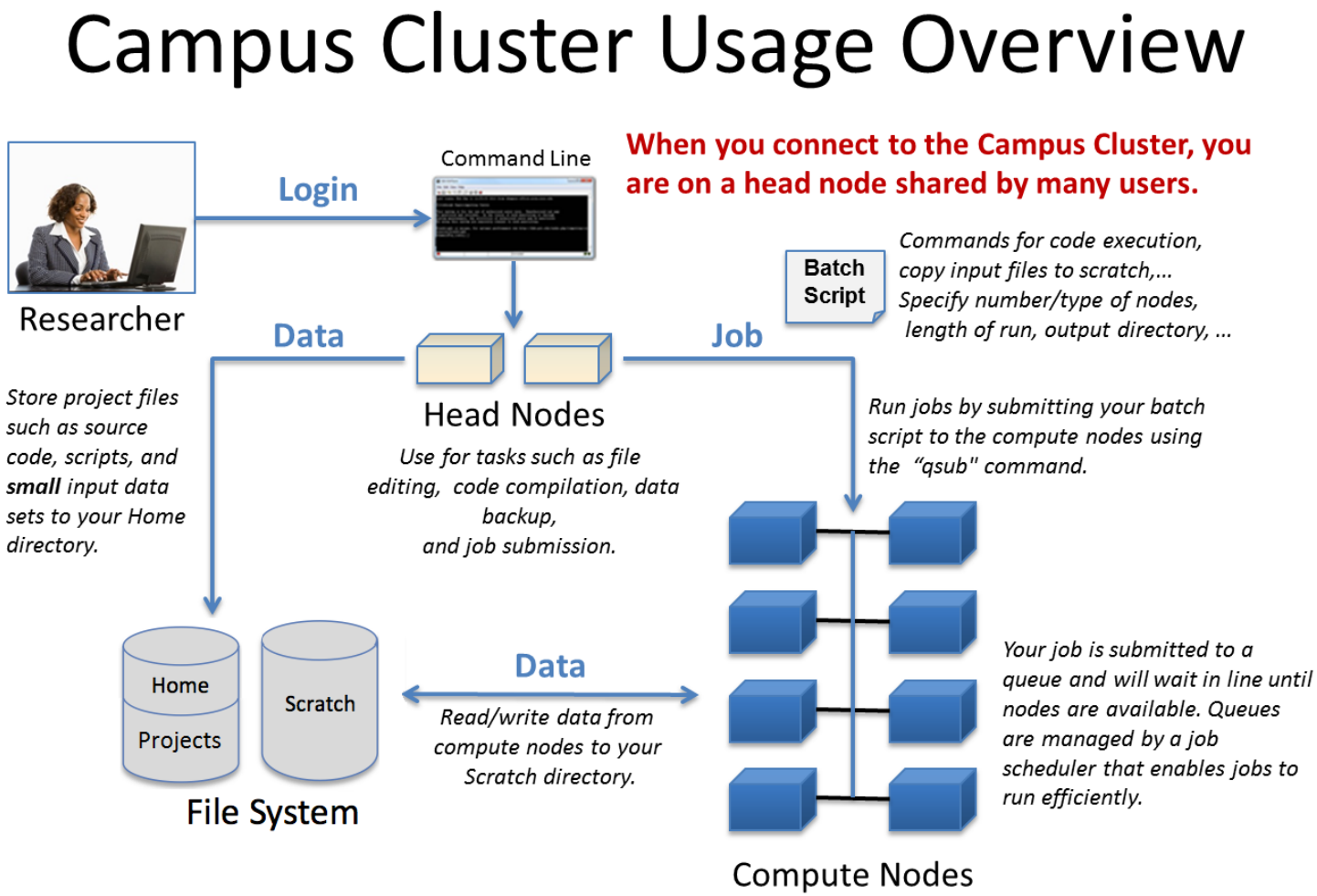
Queue Details

- The *secondary queue*, called `secondary`, allows for:
  - up to 208 nodes (on either Taub or Golub)
  - a **maximum job runtime (walltime) of 4 hours**.
- The *Statistics department* queue, `stat`, has:
  - **eight nodes (~160 cores available)**
    - 4: each with 128G of memory and 16 cores (older)
    - 4: each with 256G of memory and 24 cores (newer).
  - a **maximum walltime of 336 hours** .
- If a Professor is affiliated with the Computational Science and Engineering (CSE) organization, you can gain access to the *cse queue* that has:
  - 288 nodes
  - a **maximum walltime of 72 hours** .

ICC Specs

- Nodes on **Golub** are configured with:
  - Two 2.5 GHz Haswell (Intel E5-2680V3) processors (12 cores each for 24 cores per node),
  - two 1 TB SATA disk drives,
  - 4x Gigabit Ethernet connection / FDR InfiniBand (Optional),
  - (Optional) 2 NVIDIA Tesla K80 GPUs, and
  - either 64, 128, or 256GB RAM depending on the owner's choice.
- Nodes on **Taub** are configured with:
  - Two Intel HP X5650 2.66Ghz 6C Processors
  - HP 160GB or 500GB 3G SATA 7.2K 3.5in QR ETY HDD
  - HP IB Enablement Kit, and
  - either 12, 24, 48, or 96GB RAM owner's choice.

Structure of ICC



Node Structure

Two types of nodes:

- **Head nodes:** Login area from your laptop/desktop and a staging area (few)
- **Compute nodes:** Nodes that handle the computation from user jobs (many)

Connecting to ICC

- To work with ICC, we first need to connect to the **head node** using **Secure Shell**, more commonly known as: `ssh`
- Example login:

```
ssh netid@cc-login.campuscluster.illinois.edu
# Enter password
```

- Mine:

```
ssh balamut2@cc-login.campuscluster.illinois.edu
# nottelling
```

Tips

Repetitively typing out:

```
ssh netid@cc-login.campuscluster.illinois.edu
# password
```

is tedious. There are two tricks that void this and also make locally launched script jobs possible.

- Public/Private keys
  - Passwordless login
- SSH Config
  - Alias connection names

Public/Private Keys

Instead of entering a password, the local computer can submit a private key to be verified by a server. This is a bit more secure and avoids the hassle of constantly typing passwords.

To generate an SSH key use:

```
bash ssh-keygen -t rsa -C "netid@illinois.edu" Enter file in which to save the key (/home/demo/.ssh/id_rsa): # [Press enter] Enter passphrase (empty for no passphrase): # Write short password
```

Once this is done, make sure to copy it over to the remote server via:

```
ssh-copy-id netid@cc-login.campuscluster.illinois.edu
```

SSH Config

Add the following to `~/.ssh/config`

```
Host icc
    HostName cc-login.campuscluster.illinois.edu
    User netid
```

Note: The above assumes a default location for an SSH key. If you have a custom SSH key location add `IdentityFile ~/.ssh/sshkeyname.key` after netid.

Using Software

Unlike a traditional desktop, you must load the different software that you wish to use into the environment via `modulefiles`. To access the modules available, use either:

```
module avail
```

or visit
https://campuscluster.illinois.edu/user_info/software.html

To load and unload modules use:

```
module load <software>    # Enable
module unload <software>  # Disable
module purge              # Removes all active modules
```

Writing a Custom Module

Sometimes you may need to compile and create your own modules. Only do this if your request to ICC's help desk (via help@campuscluster.illinois.edu) goes unanswered.

A tutorial showing how to install *R* and its dependencies is available at:

http://thecoatlessprofessor.com/programming/r/a-modulefile-approach-to-compiling-r-on-a-cluster/

Setting up ICC for *R*

```
# Create a directory for your R packages
# Note: This counts against your 2 GB home dir limit on ICC
mkdir ~/Rlibs

# Load the R modulefile
# You may want to specify version e.g. R/3.2.2
module load R

# Set the R library environment variable (R_LIBS) to include
export R_LIBS=~/Rlibs

# See the path
echo $R_LIBS
```

- Always load *R* via `module load`. Otherwise, *R* will **not** be available.

Permanently setup *R* home library

- To ensure that the `R_LIBS` variable remains set even after logging out run the following command to permanently add it to the environment
  - e.g. this modifies your the `.bashrc` file, which is loaded on startup.

```
cat <<EOF >> ~/.bashrc
  if [ -n $R_LIBS ]; then
      export R_LIBS=~/Rlibs:$R_LIBS
  else
      export R_LIBS=~/Rlibs
  fi
EOF
```

Install *R* packages into home library

```
# Use the install.packages function to install your R package
$ Rscript -e "install.packages('devtools',
              '~/Rlibs', 'http://ftp.ussg.iu.edu/CRAN/')"

# Use devtools to install package
$ Rscript -e "devtools::install_github('coatless/visualize')"

# Devtools install from secret repo
$ Rscript -e "devtools::install_github('stat385uiuc/netid',
                                       subdir='secretpkg',
                                       auth_token = 'abc')"
```

- Watch the use of ' and "!
- For auth_token obtain a **GitHub Personal Access Token**

Transforming Data to and Fro ICC

- Within `bash`, there exists **Secure Copy** or `scp` that enables the transfer of files to ICC.

```
# Transferring a file on your local system to your
# home directory on the Campus Cluster:
[user@local ~]$
scp local.txt My_NetID@cc-login....edu:~/
```

```
# Transferring a file in your home directory on the
# Campus Cluster to your local system:
[user@local ~]$
scp My_NetID@cc-login....edu:~/remote.txt ./
```

- **Note:** To transfer an entire folder use: `scp -r`
- Full URL is: `cc-login.campuscluster.illinois.edu`
- See **Graphical Upload Guide** for an alternative.

Simulating n obs from $N(\mu, 1)$

- To motivate the cluster usage, we'll opt for a straightforward example.
- The goal is to be able to simulate different number of observations n from a Normal Distribution with parameters μ and $\sigma^2 = 1$.
- The exercise in itself could easily be condensed into the following short *R* script:

```
n = 20                    # Same 20
mu = 5                    # Mean of 5
set.seed(111)             # Set seed for reproducibility
rnorm(n, mean = mu) # Generate Observations
```

Understanding a Job on ICC

- In the simpliest job, there are only two "working" parts:
  - `sim_runner.R`: Script governing the desired computations.
  - `sim_job.pbs`: Controls how the job is executed on the cluster
- This setup assumes that you have no external data file to be read in or specific parameter configurations to test.

Writing sim_runner.R

- Place `sim_runner.R` in your home directory `~/`

```
# Expect command line args at the end.
args = commandArgs(trailingOnly = TRUE)

# Skip args[1] to prevent getting --args

# Extract and cast as numeric from character
rnorm(n = as.numeric(args[2]), mean = as.numeric(args[3]))
```

- Would you be able to reproduce the results?

Writing a PBS File `sim_job.pbs`: Part 1

- Modify `ppn` to increase the number of cpus if using parallelization.

```
#!/bin/bash
#
## Set the maximum amount of runtime to 4 Hours
#PBS -l walltime=04:00:00
## Request one node with `nodes` and one core with `ppn`
#PBS -l nodes=1:ppn=1
#PBS -l naccesspolicy=singleuser
## Name the job
#PBS -N job name
## Queue in the secondary queue
#PBS -q secondary
## Merge standard output into error output
#PBS -j oe
#####################################
```

Writing a PBS File sim_job.pbs: Part 2

```
## Grab the job id from an environment variable
## and create a directory for the data output
export JOBID=`echo "$PBS_JOBID" | cut -d"[" -f1`
mkdir $PBS_O_WORKDIR/"$JOBID"

## Switch directory into job ID (puts all output here)
cd $PBS_O_WORKDIR/"$JOBID"

# Load R
module load R

## Run R script in batch mode without file output
Rscript $HOME/sim_runner.R --args 5 10
```

- Setup a working directory and call sim_runner.R file,

Run the job!

- Submit your job using `qsub`

```
qsub sim_job.pbs
```

- Check job status with `qstat`

```
qstat -u netid
```

- Or visit the **Campus Cluster Status** page.

Using an Array Job

- Previously, we only ran one job with one reptition.
- In practice, we may want to run multiple reptitions across different seeds to evaluate stability or try a combination of different parameters.
- As a result, it would be highly inefficient if we constantly updated and submitted a job runner file (e.g. `sim_runner.R`) with each value.
- Instead, we opt to use something called an Array Job that allows us to submit multiple jobs.

Understanding an Array Job on ICC

- For an Array Job, there are three important parts:
  - `inputs.txt`: List of parameter values to use.
  - `sim_runner_array.R`: Script governing the desired computations.
  - `sim_array_job.pbs`: Controls how the job is executed on the cluster
- Note: We only added `inputs.txt` vs. the standard job configuration.

Modification to enable job array in .pbs file

- To enable a job array, add the following into the top of the .pbs file:

```
## Run with job array indices 1 through 6.
#PBS -t 1-6
```

- These indices are used below to get the right lines from the input file

Modification to `.pbs` file

- Change step size with `:n`, e.g.

```
#PBS -t 1-10:2
## gives 1,3,5,7,9
```

Array Job PBS File sim_array_job.pbs: Part 1

```
#!/bin/bash
#
## Set the maximum amount of runtime to 4 Hours
#PBS -l walltime=04:00:00
## Request one node with `nodes` and one core with `ppn`
#PBS -l nodes=1:ppn=1
#PBS -l naccesspolicy=singleuser
## Name the job
#PBS -N job name
## Queue in the secondary queue
#PBS -q secondary
## Run with job array indices 1 through 6.
#PBS -t 1-6
## Merge standard output into error output
#PBS -j oe
#################################################
```

Array Job PBS File sim_array_job.pbs: Part 2

```
export JOBID=`echo "$PBS_JOBID" | cut -d"[" -f1`
mkdir $PBS_O_WORKDIR/"$JOBID"

cd $PBS_O_WORKDIR/"$JOBID"

module load R

## Grab the appropriate line from the input file.
## Put that in a shell variable named "PARAMS"
export PARAMS=`cat ${HOME}/inputs.txt |
               sed -n ${PBS_ARRAYID}p`

## Run R script based on the array number.
Rscript $HOME/sim_job.R --args $PARAMS
```

Customize the parameters with an `input.txt` file.

- To customize the job, opt for an `input.txt` file via:

```
0 1
2 3.3
9 2.3
.. ..
42 4.8
```

- Note: Each line corresponds to an array ID!!

Job Array - sim_runner_array.R

```r
# Expect command line args at the end.
args = commandArgs(trailingOnly = TRUE)

# Skip args[1] to prevent getting --args

# Obtain the ID being accessed from the array
jobid = as.integer(Sys.getenv("PBS_ARRAYID"))

# Set seed for reproducibility
set.seed(jobid)

# Extract and cast as numeric from character
rnorm(n = as.numeric(args[2]), mean = as.numeric(args[3]))
```

Misc: Lots of ways to structure input args

In addition to `Base R`, there are many different options on CRAN to create correctly structured file inputs.

- getopt
- optparse
- argparse
- docopt
- argparser
- minimist
- optigrab