# Distributed Computing with ICC

James Joseph Balamuta

Departments of Informatics and Statistics
University of Illinois at Urbana-Champaign

May 03, 2018

## On the Agenda

## Overview of Distributed Computing at UIUC

## Available Clusters

- **Illinois Campus Cluster (ICC)**: Follows a time share model with a majority of departments buying in and is also usable for classes. *Try to use this option first*.
- **Keeling** (formerly **manabe**): LAS machine for faculty & graduate students. Provides a stepping stone environment to ICC usage.
- **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 under a Research Computing as a Service paradigm.
- **BlueWaters**: Expensive, but grants can be had if faculty are affiliated with NCSA. Requires two-factor authentication.

## What is ICC?

- Illinois Campus Cluster (ICC) is the public facing name to the underlying node arrangement called: Golub (deployed 2013).
    - **Note:** *Taub* is no longer functional.
- The cluster has **170+ 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

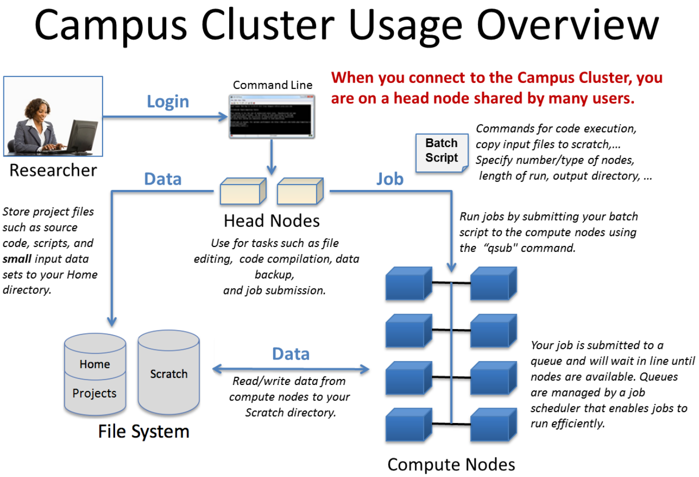
- The *secondary queue*, called `secondary`, allows for:
  - up to 208 nodes
  - a **maximum job runtime (walltime) of 4 hours**.
- The *Statistics department* queue, `stat`, has:
  - **10 nodes (~160 cores available)**
    - 4: each with 128G of memory and 16 cores (older)
    - 4: each with 256G of memory and 24 cores (newer).
    - 2: each with 256G of memory, 24 cores, and GPUs (newest).
  - 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.

## Structure of ICC


Campus Cluster Usage Overview
Researcher
Login
Command Line
When you connect to the Campus Cluster, you are on a head node shared by many users.
Batch Script
Commands for code execution, copy input files to scratch,... Specify number/type of nodes, length of run, output directory, ...
Data
Job
Head Nodes
Use for tasks such as file editing, code compilation, data backup, and job submission.
Store project files such as source code, scripts, and **small** input data sets to your Home directory.
Run jobs by submitting your batch script to the compute nodes using the "qsub" command.
Home
Projects
Scratch
File System
Data
Read/write data from compute nodes to your Scratch directory.
Your job is submitted to a queue and will wait in line until nodes are available. Queues are managed by a job scheduler that enables jobs to run efficiently.
Compute Nodes

## 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)

## Remote Access

## 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
```

## Advanced Remote Access

## 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:

```
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, copy it over to the remote server via:

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

## SSH Config[1]

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

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

[1]**Note:** This assumes a default location is used for the SSH key. If there is a custom SSH key location add `IdentityFile ~/.ssh/sshkeyname.key` after the `User` line.

## Software Modules on ICC

## 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:

```
module avail
```

or visit ICC: Software List.

To see active modules in the user space or to load and unload modules use:

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

## Latest Version of *R*

As of November 2017, the latest version of *R* on ICC is *R* 3.4.2. It is *highly* suggested that you use this version of *R* as it is compiled against a more modern *gcc* compiler (gcc 6.2.0) that: 1. fully supports OpenMP, 2. modern C++ standards, and 3. enhanced code optimization.

To load the library use:

```
module load R/3.4.2
```

If the version is not specified during the load, e.g. `module load R`, the default *R* version of 3.2.2 will be used.

## Writing a Custom Module

ICC's help desk (via `help@campuscluster.illinois.edu`) can help install software on ICC. Please send them an e-mail and *CC* either a professor or advisor.

However, if they are unable to help, it is possible to compile and create your own modules.

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/`

## User Profiles

## Setting up ICC for *R*[2]

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

# Load the R 3.4.2
module load R/3.4.2

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

# See the path
echo $R_LIBS
```

[2]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.[3]

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

[3] The routine modifies the .bashrc file, which is loaded on startup.

## 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('stat385/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.

## Single Run Job

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

- To motivate computing on a cluster, we'll opt to simulate a different number of observations $n$ from a Normal Distribution with parameters $\mu$ and $\sigma^2 = 1$.
- The exercise in itself could be condensed into the following short *R* script:

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

## Understanding a Job on ICC[4]

In a single job scenario, 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

[4] **Disclaimer:** 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 ~/

```r
# 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`: Configuring Settings[5]

```
#!/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=shared
## Name the job
#PBS -N job name
## Queue in the secondary queue
#PBS -q secondary
## Merge standard output into error output
#PBS -j oe
######################################
```

[5] Modify `ppn` to increase the number of cpus if using parallelization.

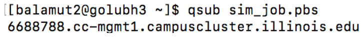
## Writing a PBS File `sim_job.pbs`: Code Execution[6]

```
## Create a directory for the data output based
## on PBS_JOBID
mkdir ${PBS_O_WORKDIR}/${PBS_JOBID}

## Switch directory into job ID (puts all output here)
cd ${PBS_O_WORKDIR}/${PBS_JOBID}

# Load R
module load R/3.4.2

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

[6]Setup a working directory and call `sim_runner.R` file.

## Run the job with qsub

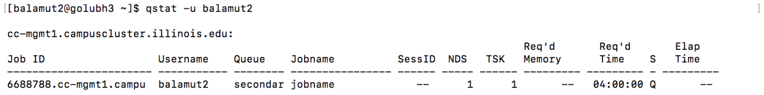
```
qsub sim_job.pbs
```



```
[balamut2@golubh3 ~]$ qsub sim_job.pbs
6688788.cc-mgmt1.campuscluster.illinois.edu
```

## Job Status Information via qstat

```
qstat -u netid
```



```
[balamut2@golubh3 ~]$ qstat -u balamut2

cc-mgmt1.campuscluster.illinois.edu:
                                                                        Req'd      Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK    Memory     Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ---------- --------- - ---------
6688788.cc-mgmt1.campu  balamut2    secondar jobname              --      1      1        --  04:00:00 Q        --
```

| Item | Description |
|---|---|
| Job ID | The job identifier assigned by PBS |
| Username | The job owner |
| Queue | The queue in which the job resides |
| Jobname | The job name given by the submitter |
| SessID | The session id (if job is running) |
| NDS | Total number of nodes |
| TSK | Total number of processors |
| Req'd Memory | The amount of memory requested by the job |
| Req'd Time | The amount of wall time used by the job |
| S | The job state |
| Elap Time | The wall time used so far |

## Job Status Codes

Common Job Status Codes: **Q**ueued, **R**unning, and **C**ompleted.

| Status Code | Description |
|---|---|
| C | Job is completed after having run |
| E | Job is exiting after having run |
| H | Job is held |
| Q | Job is queued, eligible to run or routed |
| R | Job is running |
| T | Job is being moved to new location |
| W | Job is waiting for its execution time (-a option) to be reached |
| S | (Unicos only) Job is suspended. |

Alternatively, view job status online via either the **System Status Page** or **Campus Cluster Monitor**.

## Array Jobs

## 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[7]

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

[7] **Note:** We only added `inputs.txt` vs. the standard job configuration.

## Customize the parameters with an `inputs.txt` file.

To customize the job, create an `inputs.txt` that specifies a configuration for each Job Array ID.[8]

```
0 1
2 3.3
9 2.3
8 2.1
.. ..
.. ..
42 4.8
```

The first column contains values for the *first* parameter and the second column is the values for the *second* parameter.

[8]Each line corresponds to an array ID!

## Constructing Simulation Configurations

Given different simulation settings like:

- n: sample size from 50 to 250
- rho: correlation in observations ($0 \leq p \leq 1$)
- iter: number of replications

All combinations can be generate using *R*'s expand.grid() and, then, written to inputs.txt

```
config = expand.grid(n    = seq(50, 250, by = 50),
                     rho  = seq(0, 1, by = 0.25),
                     iter = seq_len(100))
write.table(config, file = "inputs.txt", row.names = FALSE)
```

## 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 3.
#PBS -t 1-3
```

These indices are used below to get the right lines from the input file, e.g. going back to the example `inputs.txt`, we would get:

```
0 1
2 3.3
9 2.3
```

## Modifying the step size

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

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

## Accessing PBS Variables in *R*

We can access the active Array Job ID bash variable `PBS_ARRAYID` in *R* by:

```
Sys.getenv("PBS_ARRAYID")
```

This is a popular way to set a seed in a Job Array setup. e.g.

```
PBS_ARRAYID = as.numeric(Sys.getenv("PBS_ARRAYID"))
set.seed(PBS_ARRAYID)
```

For a complete list of environment variables, see PBS Environment Variables.

## 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=shared
## Name the job
#PBS -N jobname
## 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 CUSTOM_JOBID=`echo "$PBS_JOBID" | cut -d"[" -f1`
mkdir ${PBS_O_WORKDIR}/${CUSTOM_JOBID}

cd ${PBS_O_WORKDIR}/${CUSTOM_JOBID}

module load R/3.4.2

## 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
```

## 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`