Lecture 18: Distributed Computing via ICC

STAT 385 - James Balamuta

August 1, 2016

On the Agenda

- Administrative Issues
  - Group Presentations tomorrow (August 2nd, 2016) during class
  - Final Report due on August 7th at 11:59 PM CST
  - HW 5 due on August 7th at 11:59 PM CST
  - Last Office Hour August 3rd at 11AM-12:30PM.
- Unix Terminal
  - Clients
  - Basic commands
- ICC
  - Connecting
  - PBS File
  - Job Runner
  - Input

Group Presentations tomorrow (August 2nd, 2016)

- There will be 15 minutes during class that is allocated to the presentation of your project.
- Please focus on the **outcomes** of the project.
- The score is based solely on how well the project outcomes and methods used are conveyed.
  - Introduce the project by providing background / motivation
  - Describe methods used
  - Show pictures of the interface or of the generated results.
  - Discuss possibilities of future work
- There should be no more than 12 slides (not including references) in the presentation.
- **E-mail me a copy of the slides in advance!**

Final Report due on August 7th at 11:59 PM CST

Please prepare the final report using the following outline:

- Introduction
- Related Work
- Method
- Results
- Discussion
- Conclusion
- References

Reports will be graded based upon:

- project outcomes,
- the execution of methods described, and
- writing quality.

Please complete the **peer evaluations of group members** as well.

Moving along. . .

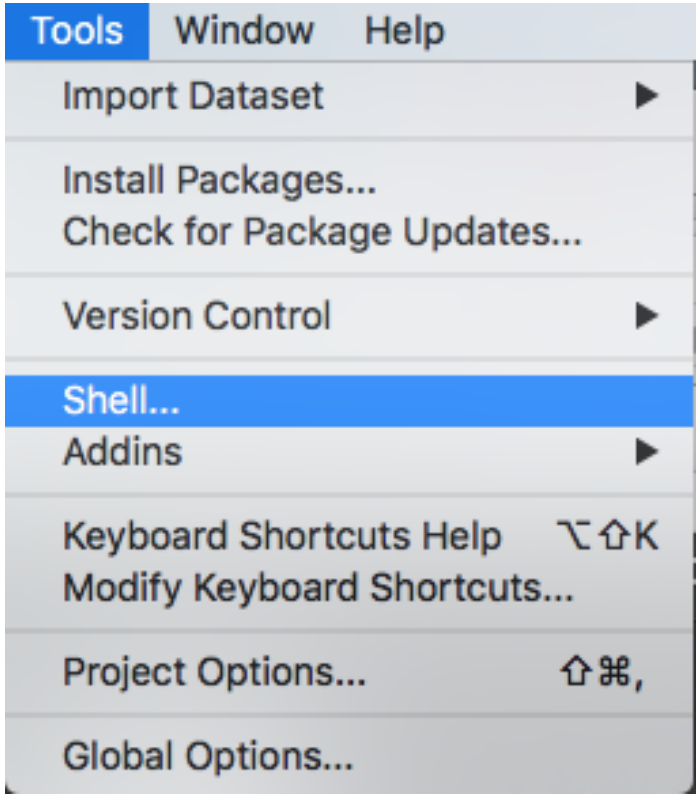
- Last administration update. Any questions?
- Moving along to. . . **Unix Shell**!

Bash (Unix Shell)

- `GNU` is a free software environment that stands for "GNU's Not Unix"
- `Bourne-Again` Shell (`bash`) written by **Brian Fox** for the GNU Project
  - Used on most Linux operating systems and on macOS.
  - Released in 1989

Accessing `bash` in *RStudio*

- By default, this is included in RStudio:
  - `Tools` ⇒ `Shell...`



- **Note:** This may not be the case on Windows as you may only receive the command line prompt.

Grab a copy of `bash`

- If on Windows, you may need to...
  - Install `git` outside of `GitHub Desktop` to have the `bash` shell.
  - **Download and Install `cygwin` on Windows**
    - Follow the windows specific instructions... Skip over the AWS parts!

Bash Prompt

When logged into `bash` it is traditional to see on the left hand side:

```
[username@hostname directory]$
```

In my case, I have:

```
[balamut2@taubh2 ~]$
```

- The ~ means "home directory" or `/home/username`

Unix Prompt commands

- **Syntax:** `command [option] [source file(s)] [target file]`
  - Options often have the `-x` or `--xxx` format
  - Use `Tab` to autocomplete source file / target file name.
- Advanced usage
  - Chain operations together via pipe operator |
  - Execute command next if previous one succeeds using &&
  - Redirect operators <, >, >>, 2> for input/output/error

Useful Unix Commands - Directories

Command	Description	Example
`pwd`	Print working directory	`pwd`
`cd`	Change directory	`cd dir/new` **or** `cd ../`
`ls`	List files	`ls ~/` **or** `ls -la new/`
`mkdir`	Make directory	`mkdir test` **or** `mkdir -p mr/r`
`rmdir`	Remove directory	`rmdir test` **or** `rmdir -p mr/r`

Unix Commands - pwd - Print working directory

```
pwd
```

```
## /Users/james/BoxSync/stat385/lectures/lec18
```

Unix Commands - cd - Change directory

```
cd ../ && pwd # Go one directory up
```

```
## /Users/james/BoxSync/stat385/lectures
```

```
cd ~/ && pwd # Go to base directory
```

```
## /Users/james
```

Unix Commands - `ls` - List Files

```
ls ~/BoxSync/stat385                    # List files
```

```
## book
## exam
## grades
## homework
## lectures
## support
## website
```

```
ls -l ~/BoxSync/stat385 | grep lec # List files with lec
```

```
## drwxr-xr-x  25 james  staff   850 Aug  1 12:19 lectures
```

Unix Commands - `mkdir` - Make Directory

- Use `mkdir` to create a new folder for a project.

```
mkdir test          # Make directory in `pwd`
```

- Adding the `-p` option allows for **all folders to be made** if not already present.

```
mkdir -p new/dir # The -p makes all directories
```

Unix Commands - `rmdir` - Remove directory

- Use `rmdir` to remove or delete a folder.

```
rmdir test          # Remove directory
```

- Including the `-p` option allows for all directory structures to be **removed**.

```
rmdir -p new/dir # The -p recursively removes
```

Useful Unix Commands - File Manipulation

Command	Description	Examples
`touch`	Make file	`touch file.R`
`vi`	Open text editor	`vi file.R`
`cat`	Display **All** of file	`cat file.R`
`chmod`	Set file permissions	`chmod 744 file.R`
`head`	Display *first* lines	`head file.R`
`tail`	Display *last* lines	`tail file.R`
`cp`	Copy file from x to y	`cp file1.R file2.R`
`mv`	Move (rename) file	`mv file_old.R file_new.R`
`rm`	Remove file	`rm file.R` **or** `rm file*.R`
`echo`	Display terminal variable	`echo $HOME`
`grep`	Regex find	`grep "toad"`

Unix Commands - touch - Touch

```
ls -l | grep "file.R" # File does not exist

# empty return
```

```
touch file.R            # Create File

ls -l | grep "file.R" # Check for existence
```

```
## -rw-r--r--  1 james  staff       0 Aug  1 14:28 file.R
```

Unix Commands - `vi` - File Editor in Terminal

```
vi file.R # Open file
```

- Navigating `vi`
  - Press `I` to insert new characters.
  - To save changes, press `Esc` and type `:w`
  - To exit, press `Esc` and type `:q!`
  - To do both at the same time use `:wq!`
- Resources:
  - **Interactive `vim` tutorial**
  - Try the **vim game** for practice
  - **`vi` Reference guide**

Note: `vim` is the sucessor to `vi` and still is applicable.

Unix Commands - Using redirection to write to file

- Redirecting output into file

```
echo "line 1" >> file.R
echo "line 2" >> file.R
```

- Using **heredoc format** to write multiple lines to file:

```
cat <<EOF >> file.R
line 3
line 4
line 5
line 6
EOF
```

- Note the following:
  - > outputs to a file
  - >> appends to a file
  - < reads input from file.

Unix Commands - cat - See file contents

```
cat file.R      # Show file contents
```

```
## line 1
## line 2
## line 3
## line 4
## line 5
## line 6
```

Unix Commands - File Permissions

- File permissions are a bit complicated but a necessary force.
- File permissions indicate whether someone can:

Type	Description	Value	Character
Execute	Run a file	1	`x`
Write	Save to a file	2	`w`
Read	See what a file contains.	4	`r`

Unix Commands - File Permissions for User Type

- Each type can be added together to customize the access level
  - For example: 7 would give all permissions, 5 gives only read and write.
- There are **three** types of permissions that can be assigned:

Type	Description	Position	Character
User	Owner or user	First	`u`
Group	Those that belong to a group	Second	`g`
World	Everyone.	Third	`a`

Unix Commands - chmod - Set File Permissions

```
chmod 777 file.R       # Everyone can read, write, access

chmod u+wxr file.R     # Only owner can read, write, access
```

Unix Commands - `head` - See top content

```
head -2 file.R    # Show top 2 lines
```

```
## line 1
## line 2
```

- The -2 limits it to the **top** 2 observations

Unix Commands - `tail` - See bottom content

```
tail -1 file.R    # Show last line
```

```
## line 6
```

- The `-1` limits it to the **last** observation

Unix Commands - cp - Copy File

```
cp file.R file.R.bck # Create a back up
```

```
ls -l | grep ".bck"  # Check that it is there
```

```
## -rw-r--r--  1 james  staff      42 Aug  1 14:28 file.R.b
```

- It is good practice to create `.bck` up files
- This is especially the case if you are working with configuration files (e.g. `.conf`)

Unix Commands - mv - Move File

```
mv file.R.bck file_in_use.R          # Rename file

mv file_in_use.R img/file_in_use.R # Move to new directory
```

Unix Commands - rm - Remove file

```
rm file.R                  # Remove file
```

```
# Remove file in different directory
rm img/file_in_use.R
```

Unix Commands - `echo` - Display `bash` variables

```
samplevar="Hi stat385"     # Create a variable

echo $samplevar            # Print variable
```

```
## Hi stat385
```

Note the following:

- No space between variable, assignment operator, and value.
- The use of $ to refer to the variable in `echo`.

Moving along...

- And that's it for **Unix Shell**, any questions?
- Onto... **Distributed Computing**!

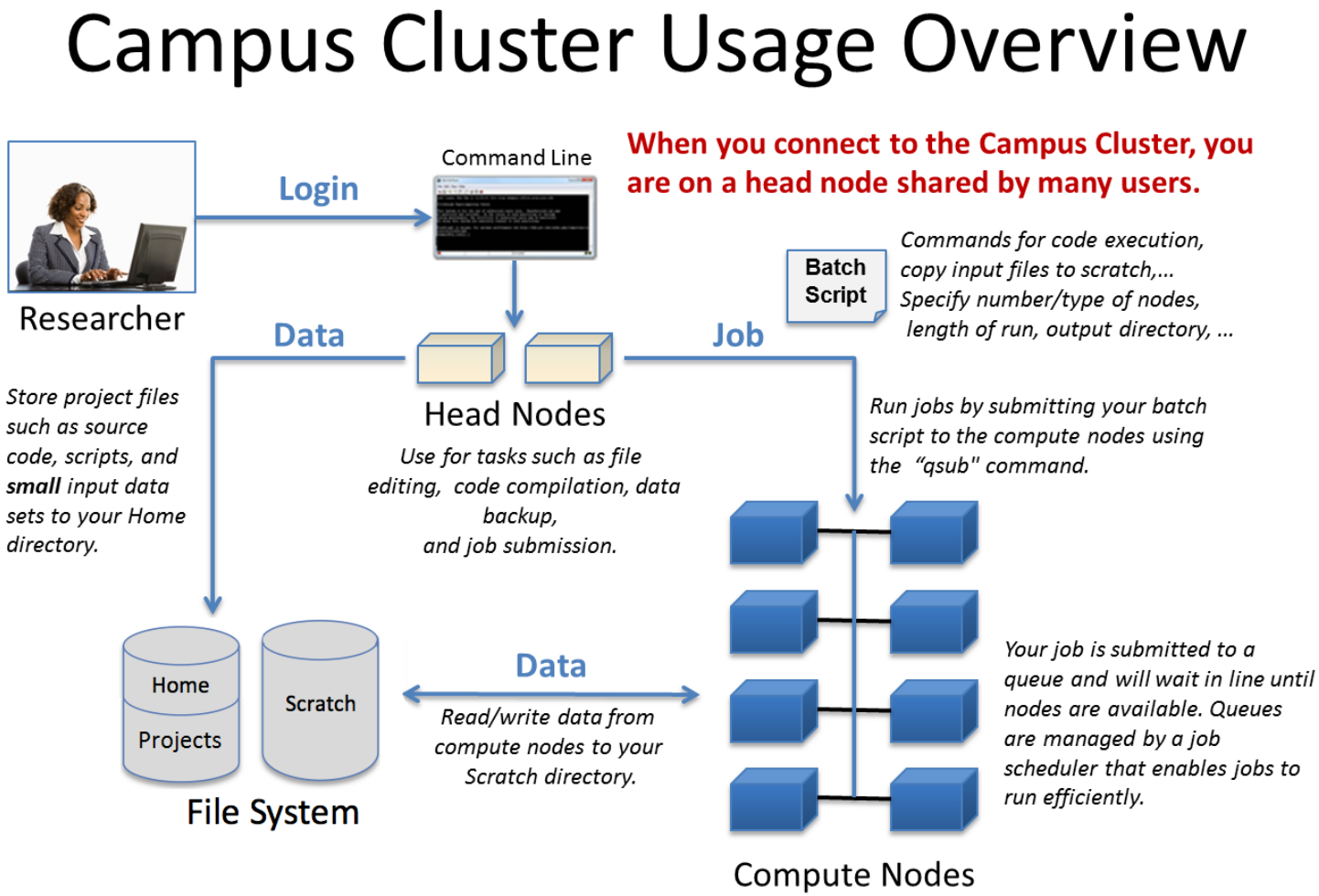
What is ICC?

- Illinois Campus Cluster (ICC), there are two unique systems: Taub and Golub. The latter is a newer deployment (2013 vs. 2011).
- There are presently about **530+ computing nodes available for use**.
- The *Statistics department* has:
  - **eight nodes (160 cores available)**
  - a **maximum job runtime (walltime) of 336 hours** in `stat` queue.
- One can submit a job on the secondary queue with a maximum walltime of 4 hours with up to 208 nodes (on either Taub or Golub).

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, 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
```

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 inclu
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 packa
$ Rscript -e "install.packages('devtools',
              '~/Rlibs', 'http://ftp.ussg.iu.edu/CRAN/')"

# Use devtools to install package
$ Rscript -e "devtools::install_github('SMAC-Group/gmwm')"

# 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.

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 `~/`

```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]))
```

Writing a PBS File `sim_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
## Merge standard output into error output
#PBS -j oe
#####################################
```

- Standard job template.
- Change the `ppn` to increase the number of processors allowed if using parallelization.

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
```

- This calls the `sim_runner.R` file and setups a working directory for the job to be run on.

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 theright 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