High Performance Computing

James Joseph Balamuta

1/6/23

Table of contents

1 Welcome to HPC

The goal of High Performance Computing on a Cluster with *R* is to inform and distribute various recipes and techniques for performing computational-intensive work on remote computing resources.

1.0.1 Thanks!

This work made use of the Illinois Campus Cluster, a computing resource that is operated by the Illinois Campus Cluster Program (ICCP) in conjunction with the National Center for Supercomputing Applications (NCSA) which is supported by funds from the University of Illinois at Urbana-Champaign.

Conversations with the ICCP staff has also greatly helped in developing material. In particular, I would like to thank:

- Weddie Jackson
- Matthew Long
- Chit Khin

Part I

Overview of Cluster Computing

2 Cluster Computing

2.1 What is Cluster Computing?

Definition: Cluster

A **cluster** is a *set of computers* that are connected together and share resources as if they were one gigantic computer.

2.2 How Does Cluster Computing WorK?

Definition: Parallel Processing

Parallel Processing is the act of carrying out multiple tasks simultaneously to solve a problem. This is accomplished by dividing the problem into independent subparts, which are then solved concurrently.

Definition: Jobs

Jobs denote the independent subparts.

2.3 Why use Cluster Computing?

Pros

- Speeds up simulations by allowing iterations to be run simultaneously.
- Provides more resources for computations.
  - e.g. CPU Cores, RAM, Hard Drive Space, and Graphics Cards (GPUs).
- Nightly snapshots/backups of files.
- Extends the lifespan of your computer.

Cons

- Simulations are **not** instantly run.
  - Need to "queue" for resources.

- Higher barrier of entry due to knowledge requirements.
- Poorly handles opening and closing data sets.
- Adding or updating software is complex.

3 Cluster Software

3.1 Software Modules

Unlike a traditional desktop, you must load the different software that you wish to use into the environment via `modulefiles`. The list of supported software can be found on Software List or by typing:

```
module avail
```

3.2 Viewing, Retrieving, and Disabling Module Software

The most frequently used module commands are:

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

Replace `<software>` with the name of the desired software module from `module avail`.

3.3 Latest Version of *R*

As of **September 2021**, the latest version of *R* on ICC is *R* **4.1.1**. We recommend using the latest version of R with the `_sandybridge` suffix. The reason for using `_sandybridge` is to ensure compatibility on older nodes inside of the `stat` partition. For an example of a compatibility error, please see (**debugging-errors?**).

Moreover, with this version, the default library does not contain any non-standard packages.

R can be accessed by using:

```
# Load software
module load R/4.1.1_sandybridge
```

Note: If the version is not specified during the load, e.g. `module load R`, then a default version of *R* will be used. This default may change without warning.

Once *R* is loaded, the Terminal/non-GUI version of *R* can be started by typing:

```
R
```

To exit an *R* session on the cluster, type inside *R*:

```
q(save = "no")
```

This will terminate the *R* session without saving any environment values.

3.4 Ask for Help

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

3.4.1 Writing a Custom Module

It is possible to compile and create your own modules. For details, see the tutorial A Modulefile Approach to Compiling *R* on a Cluster.

4 Storage

For additional details related to the illinois campus cluster, please the Storage and Data Guide

4.1 Storing Data & Code

- Home Directory `~/`
  - Up to ~**5GB** (Soft cap) / ~**7GB** (Hard cap) with **nightly backups**.
  - Storage is **private**.
- Project Spaces `/projects/stat/shared/$USER`
  - ~**21TB** of shared space with **nightly backups**.
  - Storage is **shared** among `stat` members.
- Temporary Networked Storage `/scratch`
  - ~**10TB** of space purged after **30 days** with **no backup**.
  - Storage is **private**.

Soft caps: gently warn the user to lower their storage size. **Hard caps**: prevent the user from adding new files.

4.2 Backups

4.2.1 Backup Info

- **Daily** night time backups.
- **30 days** of backups exist.
- **No off-site backups for disaster recovery.**

4.2.2 Location of Backups

- Home Directory `~/`

```
/gpfs/iccp/home/.snapshots/home_YYYYMMDD*/$USER
```

- Project Directory `/projects/stat/shared/$USER`

```
/gpfs/iccp/projects/stat/.snapshots/statistics_YYYYMMDD*
```

5 Cluster Setup

Within this chapter, we will cover establishing a workspace on the Campus Cluster. Workspace setup usually requires about 5 different steps.

- Ensure the cluster can easily be accessed from a local computer.
- Enable command shortcuts through aliases.
- Setup a GitHub access token for pulling software in from private repositories (skip if not needed).
- Create a space on a project drive for where R packages should be installed.
- Install *R* packages!

5.1 Secure Shell (SSH) Setup

For accessing a cluster from command line, **Secure Shell (SSH)** is preferred. Access to the cluster requires typing out each time:

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

Connecting in this manner is tedious since it involves repetitively typing out login credentials. There are two tricks that void the necessity to do so. Effectively, we have:

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

Thus, instead of entering a password, the local computer can submit a private key to be verified by a server. Not only is this more secure, but it avoids the hassle of remembering the password and typing it out while observers watch. Secondly, the connection alias will allow for typing:

```
ssh icc
```

Not bad eh?

5.1.1 Generating an SSH Key

On your **local** computer, open up Terminal and type:

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

5.1.2 Copy SSH Key to Server

Next, let's copy the generated key from your **local** computer onto the cluster.

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

On macOS, prior to using `ssh-copy-id`, the command will need to be installed. Homebrew provides a formula that will setup the command. Install using:

```
# Install homebrew
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/insta
# Install the required command binary
brew install ssh-copy-id
```

5.1.3 SSH Config File

Inside of `~/.ssh/config`, add the following host configuration. Make sure to **replace** `<netid>` value with your personal netid.

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

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.