Lecture 9: Cluster computing

Brian Williamson

BIOST 561: Computational Skills For Biostatistics I

29 May 2019

Motivation

We are often interested in large-scale computing:

Motivation

We are often interested in large-scale computing:

- simulation studies (e.g., lecture 4)

Motivation

We are often interested in large-scale computing:

- simulation studies (e.g., lecture 4)
- intensive data analyses (e.g., air pollution and mortality)

Motivation

We are often interested in large-scale computing:

- simulation studies (e.g., lecture 4)
- intensive data analyses (e.g., air pollution and mortality)

In Lecture 8, you learned how to compute without `R`.

Today, you'll learn how to transfer those skills to computing on a cluster.

Examples from my research

Large-scale simulation studies:

Examples from my research

Large-scale simulation studies:

- proof-of-concept examples

Examples from my research

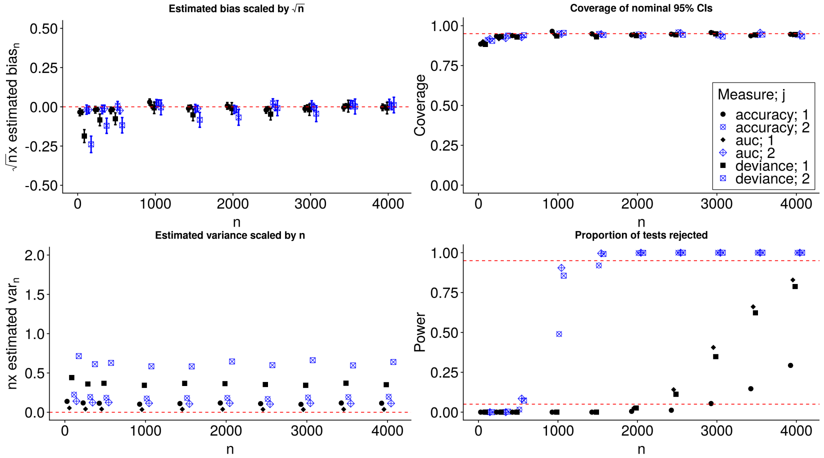
Large-scale simulation studies:

- proof-of-concept examples
- showcase operating characteristics of proposed method

Examples from my research

Large-scale simulation studies:

- proof-of-concept examples
- showcase operating characteristics of proposed method



Examples from my research

Data analysis:

Examples from my research

Data analysis:

- fit a time-consuming estimator

Examples from my research

Data analysis:

- fit a time-consuming estimator
- cross-validation?

Examples from my research

Data analysis:

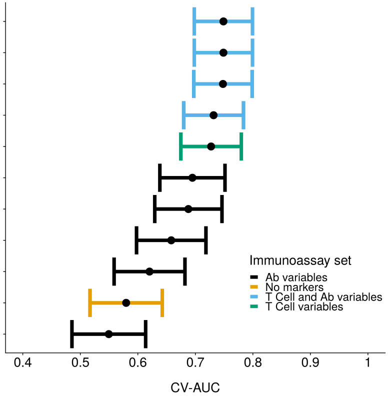
- fit a time-consuming estimator
- cross-validation?
- memory-intensive computations?

Examples from my research

Data analysis:

- fit a time-consuming estimator
- cross-validation?
- memory-intensive computations?

Assay combination	CV-AUC [95% CI]
IgG + IgA + IgG3 + T Cells	0.749 [0.698, 0.799]
IgG + IgA + T Cells	0.749 [0.698, 0.799]
All markers	0.748 [0.697, 0.799]
T Cells + Fx Ab	0.732 [0.680, 0.784]
T Cells	0.727 [0.675, 0.780]
IgG3	0.695 [0.638, 0.751]
IgG + IgA + IgG3	0.688 [0.629, 0.746]
IgG + IgA + IgG3 + Fx Ab	0.658 [0.598, 0.718]
IgG + IgA	0.620 [0.559, 0.682]
No markers	0.580 [0.517, 0.643]
Fx Ab	0.550 [0.485, 0.614]



The problem

How do I run

The problem

How do I run

- many

The problem

How do I run

- many
- potentially time-consuming

The problem

How do I run

- many
- potentially time-consuming
- or memory-intensive

The problem

How do I run

- many
- potentially time-consuming
- or memory-intensive

analyses without

The problem

How do I run

- many
- potentially time-consuming
- or memory-intensive

analyses without keeping an R session open (for the duration!)

The problem

How do I run

- many
- potentially time-consuming
- or memory-intensive

analyses without keeping an R session open (for the duration!) on my personal machine?

The solution

Cluster computers provide a solution!

The solution

Cluster computers provide a solution!

Cluster computers:

The solution

Cluster computers provide a solution!

Cluster computers:

- allow you to submit multiple **jobs*** at once

The solution

Cluster computers provide a solution!

Cluster computers:

- allow you to submit multiple **jobs*** at once
- can schedule jobs for you

The solution

Cluster computers provide a solution!

Cluster computers:

- allow you to submit multiple **jobs*** at once
- can schedule jobs for you
- are optimized for high-performance computing (HPC)

The solution

Cluster computers provide a solution!

Cluster computers:

- allow you to submit multiple **jobs*** at once
- can schedule jobs for you
- are optimized for high-performance computing (HPC)

*: we will discuss this further!

Running example: robust standard errors (SEs)

Consider a sample of n observations generated iid according to

$$\begin{aligned} X &\sim N(0,1), \\ u &\sim N(0,1), \text{ independent of } X; \\ Y \mid X, u &= \beta_0 + \beta_1 X + \epsilon, \text{ where} \\ \epsilon &= |X| u. \end{aligned}$$

Questions: can we use

1. linear regression to estimate β_1?

Running example: robust standard errors (SEs)

Consider a sample of n observations generated iid according to

$$\begin{aligned} X &\sim N(0,1), \\ u &\sim N(0,1), \text{ independent of } X; \\ Y \mid X, u &= \beta_0 + \beta_1 X + \epsilon, \text{ where} \\ \epsilon &= |X| u. \end{aligned}$$

Questions: can we use

1. linear regression to estimate β_1? Yes!

Running example: robust standard errors (SEs)

Consider a sample of n observations generated iid according to

$$\begin{aligned} X &\sim N(0,1), \\ u &\sim N(0,1), \text{ independent of } X; \\ Y \mid X, u &= \beta_0 + \beta_1 X + \epsilon, \text{ where} \\ \epsilon &= |X| u. \end{aligned}$$

Questions: can we use

1. linear regression to estimate β_1? Yes!
2. model-based standard errors to estimate $sd(\beta_1)$?

Running example: robust standard errors (SEs)

Consider a sample of n observations generated iid according to

$$\begin{aligned} X &\sim N(0,1), \\ u &\sim N(0,1), \text{ independent of } X; \\ Y \mid X, u &= \beta_0 + \beta_1 X + \epsilon, \text{ where} \\ \epsilon &= |X| u. \end{aligned}$$

Questions: can we use

1. linear regression to estimate β_1? Yes!
2. model-based standard errors to estimate $sd(\beta_1)$? No!

Running example: robust standard errors (SEs)

Consider a sample of n observations generated iid according to

$$\begin{aligned} X &\sim N(0,1), \\ u &\sim N(0,1), \text{ independent of } X; \\ Y \mid X, u &= \beta_0 + \beta_1 X + \epsilon, \text{ where} \\ \epsilon &= |X| u. \end{aligned}$$

Questions: can we use

1. linear regression to estimate β_1? Yes!
2. model-based standard errors to estimate $sd(\beta_1)$? No!
3. robust standard errors to estimate $sd(\beta_1)$?

Running example: robust standard errors (SEs)

Consider a sample of n observations generated iid according to

$$\begin{aligned} X &\sim N(0,1), \\ u &\sim N(0,1), \text{ independent of } X; \\ Y \mid X, u &= \beta_0 + \beta_1 X + \epsilon, \text{ where} \\ \epsilon &= |X| u. \end{aligned}$$

Questions: can we use

1. linear regression to estimate β_1? Yes!
2. model-based standard errors to estimate $sd(\beta_1)$? No!
3. robust standard errors to estimate $sd(\beta_1)$? Yes!

Running example: robust SEs

Goals:

Running example: robust SEs

Goals:

- compare model-based to robust SEs

Running example: robust SEs

Goals:

- compare model-based to robust SEs
- do this without using our own computers

Running example: robust SEs

Goals:

- compare model-based to robust SEs
- do this without using our own computers

We will use the cluster to do this!

Part I: coding for the cluster

Coding for the cluster

Before moving to the cluster, we need to code differently:

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output (for results!)

The `simulator` (Lecture 4) makes this easy:

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output (for results!)

The `simulator` (Lecture 4) makes this easy:

- forces you to code modularly

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output (for results!)

The `simulator` (Lecture 4) makes this easy:

- forces you to code modularly
- forces you to set a seed

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output (for results!)

The `simulator` (Lecture 4) makes this easy:

- forces you to code modularly
- forces you to set a seed
- forces you to save lots of output

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output (for results!)

The `simulator` (Lecture 4) makes this easy:

- forces you to code modularly
- forces you to set a seed
- forces you to save lots of output

A couple of drawbacks:

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output (for results!)

The `simulator` (Lecture 4) makes this easy:

- forces you to code modularly
- forces you to set a seed
- forces you to save lots of output

A couple of drawbacks:

- can cause memory leaks

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output (for results!)

The `simulator` (Lecture 4) makes this easy:

- forces you to code modularly
- forces you to set a seed
- forces you to save lots of output

A couple of drawbacks:

- can cause memory leaks
- sometimes difficult to link to cluster nodes

Coding for the cluster

Before moving to the cluster, we need to code differently:

- modular code (easy debugging)
- setting seeds (reproducible results)
- saving output (for results!)

The `simulator` (Lecture 4) makes this easy:

- forces you to code modularly
- forces you to set a seed
- forces you to save lots of output

A couple of drawbacks:

- can cause memory leaks
- sometimes difficult to link to cluster nodes

Today, I'll provide an alternative method (this should not always replace the simulator!).

Coding for the cluster: modular code

Modular code: each file has a single task

Coding for the cluster: modular code

Modular code: each file has a single task

Your turn!

Go to the `code` subdirectory. Then answer these questions **with a partner**:

1. what does `do_one` do? What are its arguments?
2. what does `generate_data` do? What are its arguments?
3. Write effective comments in each file so that **future you** understands each!

Coding for the cluster: modular code

Answers and Brian's comments (to be posted after class)

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

- generating data (e.g., simulation)

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

- generating data (e.g., simulation)
- cross-validation

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

- generating data (e.g., simulation)
- cross-validation
- ...

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

- generating data (e.g., simulation)
- cross-validation
- ...

and you want it to be reproducible (you do!).

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

- generating data (e.g., simulation)
- cross-validation
- ...

and you want it to be reproducible (you do!).

Things to remember on setting a seed:

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

- generating data (e.g., simulation)
- cross-validation
- ...

and you want it to be reproducible (you do!).

Things to remember on setting a seed:

- set seeds in a logical, reproducible way

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

- generating data (e.g., simulation)
- cross-validation
- ...

and you want it to be reproducible (you do!).

Things to remember on setting a seed:

- set seeds in a logical, reproducible way
- set a unique seed for each job

Coding for the cluster: setting the seed

Setting a seed is vital if your code involves

- generating data (e.g., simulation)
- cross-validation
- . . .

and you want it to be reproducible (you do!).

Things to remember on setting a seed:

- set seeds in a logical, reproducible way
- set a unique seed for each job

Coding for the cluster: #! and executables

In Lecture 8, you learned about #! and executable files.

Coding for the cluster: #! and executables

In Lecture 8, you learned about #! and executable files.

#! ("shebang"):

Coding for the cluster: #! and executables

In Lecture 8, you learned about `#!` and executable files.

`#!` ("shebang"):

- tells your computer where to look for executable file to run your code
- place at top of file

Coding for the cluster: #! and executables

In Lecture 8, you learned about #! and executable files.

#! ("shebang"):

- tells your computer where to look for executable file to run your code
- place at top of file

Executables (e.g., /bin/sh):

Coding for the cluster: `#!` and executables

In Lecture 8, you learned about `#!` and executable files.

`#!` ("shebang"):

- tells your computer where to look for executable file to run your code
- place at top of file

Executables (e.g., `/bin/sh`):

- actually run your code
- other examples: `/bin/bash`, `/usr/local/bin/Rscript`, `/usr/local/bin/python3`

Coding for the cluster: saving output

Running without a graphical user interface (GUI) requires you to pre-specify the output you want to save.

How much/what to save?

Coding for the cluster: saving output

Running without a graphical user interface (GUI) requires you to pre-specify the output you want to save.

How much/what to save? Depends on what you need!

Coding for the cluster: saving output

Running without a graphical user interface (GUI) requires you to pre-specify the output you want to save.

How much/what to save? Depends on what you need!

- More output: easy debugging, wastes memory/time

Coding for the cluster: saving output

Running without a graphical user interface (GUI) requires you to pre-specify the output you want to save.

How much/what to save? Depends on what you need!

- More output: easy debugging, wastes memory/time
- Think ahead: careful planning can make results tidy/nice

Coding for the cluster: saving output

Running without a graphical user interface (GUI) requires you to pre-specify the output you want to save.

How much/what to save? Depends on what you need!

- More output: easy debugging, wastes memory/time
- Think ahead: careful planning can make results tidy/nice

This is important regardless of whether or not you use the `simulator`!

Handy functions for saving output: `saveRDS`.

Coding for the cluster: saving output

Your turn!

In the robust SEs example, our goal is to compare model-based to robust SEs. **With a partner**, answer the following questions:

1. What output should we save?
2. How should we evaluate performance of the SEs?

Coding for the cluster: robust SEs

Check out `run_sim_robust_se.R` in the `code` directory!

Coding for the cluster: debugging

Debugging is hard without a GUI.

Coding for the cluster: debugging

Debugging is hard without a GUI.

Best practices (in my opinion):

Coding for the cluster: debugging

Debugging is hard without a GUI.

Best practices (in my opinion):

- debug **everything** on your machine first

Coding for the cluster: debugging

Debugging is hard without a GUI.

Best practices (in my opinion):

- debug **everything** on your machine first
- modular code

Coding for the cluster: debugging

Debugging is hard without a GUI.

Best practices (in my opinion):

- debug **everything** on your machine first
- modular code (isolate bugs)

Coding for the cluster: debugging

Debugging is hard without a GUI.

Best practices (in my opinion):

- debug **everything** on your machine first
- modular code (isolate bugs)
- run one job for subset of parameters prior to cluster

Coding for the cluster: compiling output

Check out `load_sim_robust_se.R` in the `code` directory!

Part II: using the cluster

Using the cluster

Department resources for HPC*:

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)
- `bayes`: compute cluster

A cluster consists of:

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)
- `bayes`: compute cluster

A cluster consists of:

- a head node

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)
- `bayes`: compute cluster

A cluster consists of:

- a head node (where you are)

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)
- `bayes`: compute cluster

A cluster consists of:

- a head node (where you are)
- compute nodes

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)
- `bayes`: compute cluster

A cluster consists of:

- a head node (where you are)
- compute nodes (where your code gets run)

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)
- `bayes`: compute cluster

A cluster consists of:

- a head node (where you are)
- compute nodes (where your code gets run)
- submission system

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)
- `bayes`: compute cluster

A cluster consists of:

- a head node (where you are)
- compute nodes (where your code gets run)
- submission system (how your code gets run)

Using the cluster

Department resources for HPC*:

- `cox`: 12-core computer (not a cluster)
- `bayes`: compute cluster

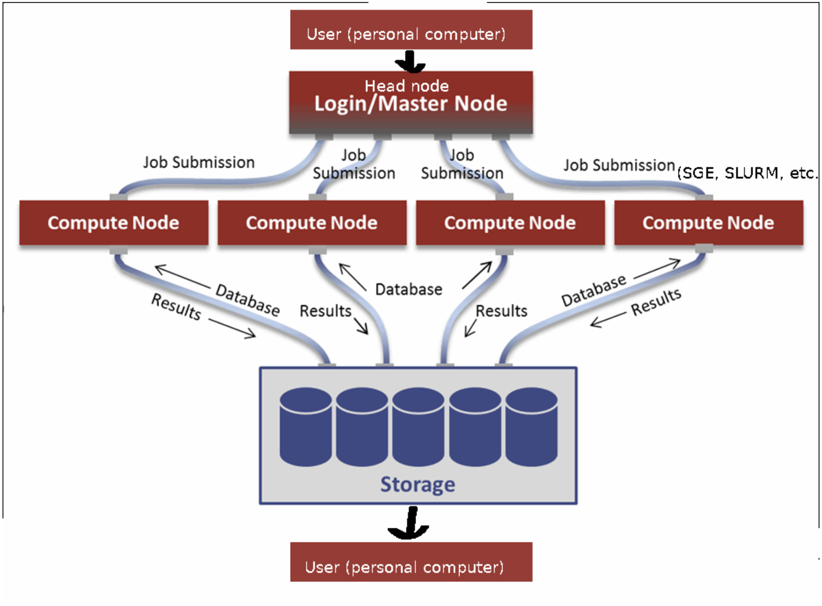
A cluster consists of:

- a head node (where you are)
- compute nodes (where your code gets run)
- submission system (how your code gets run)

*other groups have similar resources, e.g.,

- `hyak` (managed by UW-IT)
- `pearson` (statistical genetics group only)
- `gizmo` (Fred Hutchinson Cancer Research Center only)
- Microsoft Azure, Amazon Web Services (AWS)

Using the cluster



Using the cluster: `bayes`

More specifically, `bayes`

Using the cluster: `bayes`

More specifically, `bayes`

- has 4 department-wide compute nodes, each with 12 cores

Using the cluster: `bayes`

More specifically, `bayes`

- has 4 department-wide compute nodes, each with 12 cores
- uses Sun Grid Engine (SGE) for submissions

Using the cluster: `bayes`

More specifically, `bayes`

- has 4 department-wide compute nodes, each with 12 cores
- uses Sun Grid Engine (SGE) for submissions

Since this is a shared resource, being nice is important:

Using the cluster: `bayes`

More specifically, `bayes`

- has 4 department-wide compute nodes, each with 12 cores
- uses Sun Grid Engine (SGE) for submissions

Since this is a shared resource, being nice is important:

- don't run simulations on the head node

Using the cluster: `bayes`

More specifically, `bayes`

- has 4 department-wide compute nodes, each with 12 cores
- uses Sun Grid Engine (SGE) for submissions

Since this is a shared resource, being nice is important:

- don't run simulations on the head node
- don't flood the cluster

Using the cluster: `bayes`

More specifically, `bayes`

- has 4 department-wide compute nodes, each with 12 cores
- uses Sun Grid Engine (SGE) for submissions

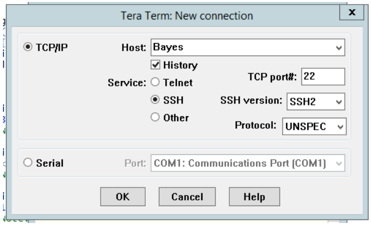
Since this is a shared resource, being nice is important:

- don't run simulations on the head node
- don't flood the cluster

We'll practice good habits for being nice as we go along.

Using the cluster: logging in (Windows, incl. box)

1. Open TeraTerm [or your favorite secure shell (SSH) client]
2. Enter the address of your favorite cluster, e.g., `bayes.biostat.washington.edu`
3. Make sure that the "New connection" window is filled out as in the figure, except for "Host" (screenshot from `box`)
4. Click `OK`, enter your UW BIOST username and password when prompted (user and pass you use for `box`)


Tera Term: New connection

[● TCP/IP] Host: Bayes
[✓] History
Service: ○ Telnet [● SSH] ○ Other
TCP port#: 22
SSH version: SSH2
Protocol: UNSPEC

○ Serial Port: COM1: Communications Port (COM1)

Using the cluster: logging in

First step: make sure you are through the department firewall.

How: OpenSesame (on biost intranet)

1. Navigate to OpenSesame website
2. Select host (e.g., `Bayes`)
3. Click `OpenSesame`

Using the cluster: logging in (Mac/Linux)

1. Open a Terminal window
2. Type `ssh mynetid@cluster.washington.edu`
   - replace `mynetid` with Your UW NetID, and
   - replace `cluster` with Your cluster, e.g., `bayes`
3. Enter password (same password as `box`) when prompted (the field will remain blank but your password will be received)

Using the cluster: logging in

Your turn! Take 2 minutes to complete this activity **by yourself** (if you don't have a computer, write down how you would do this).

1. Log into `bayes`
2. What is the name of the directory that you are when you log in?
3. Create a directory called `robust_ses` in your `home` directory
4. Create a directory called `robust_ses` on **your** computer, under `biost561/lecture9`

Using the cluster: moving around

The cluster runs on Linux: in particular, the tools you learned in lecture 8, including

Using the cluster: moving around

The cluster runs on Linux: in particular, the tools you learned in lecture 8, including

- navigation,

Using the cluster: moving around

The cluster runs on Linux: in particular, the tools you learned in lecture 8, including

- navigation,
- vim,

Using the cluster: moving around

The cluster runs on Linux: in particular, the tools you learned in lecture 8, including

- navigation,
- vim,
- commands,

Using the cluster: moving around

The cluster runs on Linux: in particular, the tools you learned in lecture 8, including

- navigation,
- vim,
- commands, and
- shell scripts

are all used in exactly the same way!

Using the cluster: jobs

The basic unit of cluster computing is a job.

Using the cluster: jobs

The basic unit of cluster computing is a job.

Jobs:

Using the cluster: jobs

The basic unit of cluster computing is a job.

Jobs:

- perform a specified task

Using the cluster: jobs

The basic unit of cluster computing is a job.

Jobs:

- perform a specified task
- can be submitted to the cluster compute nodes

Using the cluster: jobs

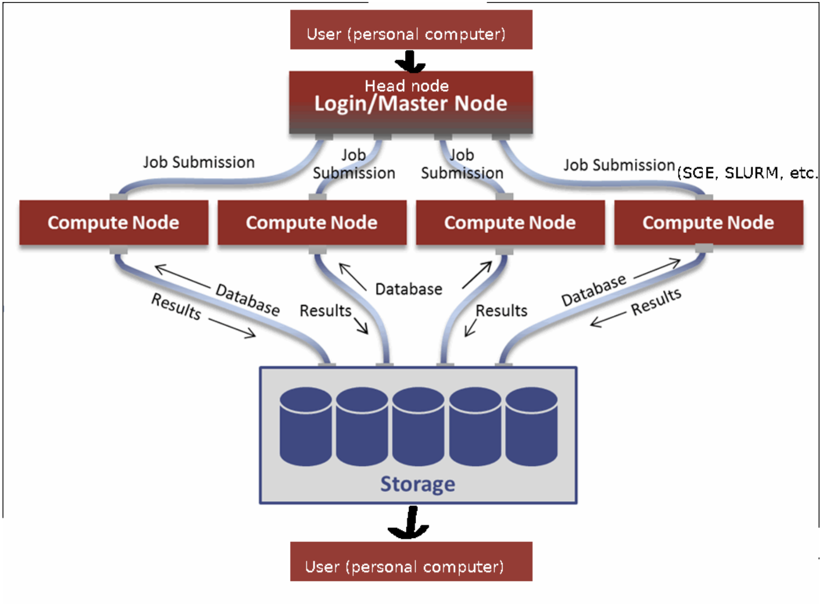
The basic unit of cluster computing is a job.

Jobs:

- perform a specified task
- can be submitted to the cluster compute nodes

Example job: run `run_sim_robust_se.R` with 50 replicates

Using the cluster: jobs



Using the cluster: calling your R script

Your turn!

Check out the file `call_sim_robust_se.sh`. **With a partner**, answer the following questions:

1. What does the `$` mean?
2. How many command-line arguments does `run_sim_robust_se.R` take?
3. What do the command-line arguments do?

Using the cluster: submitting jobs

Workhorse command on SGE: qsub

Many options, including:

Using the cluster: submitting jobs

Workhorse command on SGE: qsub

Many options, including:

- `cwd`: execute script in current working directory

Using the cluster: submitting jobs

Workhorse command on SGE: qsub

Many options, including:

- `cwd`: execute script in current working directory
- `e` and `o`: send error and output files to a folder, e.g., `iotrash/`

Using the cluster: submitting jobs

Workhorse command on SGE: qsub

Many options, including:

- `cwd`: execute script in current working directory
- `e` and `o`: send error and output files to a folder, e.g., `iotrash/`
- `t <task1-taskn>` submits a job array with `taskn`−`task1` tasks

Using the cluster: submitting jobs

Workhorse command on SGE: qsub

Many options, including:

- `cwd`: execute script in current working directory
- `e` and `o`: send error and output files to a folder, e.g., `iotrash/`
- `t <task1-taskn>` submits a job array with `taskn`−`task1` tasks (more to come on this)

Options are specified with a single -, as you saw in Lecture 8.

Using the cluster: job arrays

Continuum of task types:

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel:

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial:

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks (e.g., `run_sim_robust_se.R` with one replicate per job)

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks (e.g., `run_sim_robust_se.R` with one replicate per job)

Job arrays make embarrassingly parallel tasks easy:

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks (e.g., `run_sim_robust_se.R` with one replicate per job)

Job arrays make embarrassingly parallel tasks easy:

- unique identifier for an entire set of jobs

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks (e.g., `run_sim_robust_se.R` with one replicate per job)

Job arrays make embarrassingly parallel tasks easy:

- unique identifier for an entire set of jobs (easy to manage)

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks (e.g., `run_sim_robust_se.R` with one replicate per job)

Job arrays make embarrassingly parallel tasks easy:

- unique identifier for an entire set of jobs (easy to manage)
- unique identifier for each task within array

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks (e.g., `run_sim_robust_se.R` with one replicate per job)

Job arrays make embarrassingly parallel tasks easy:

- unique identifier for an entire set of jobs (easy to manage)
- unique identifier for each task within array (set simulation parameters)

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks (e.g., `run_sim_robust_se.R` with one replicate per job)

Job arrays make embarrassingly parallel tasks easy:

- unique identifier for an entire set of jobs (easy to manage)
- unique identifier for each task within array (set simulation parameters)
- displays nicely on cluster

Using the cluster: job arrays

Continuum of task types:

- embarrassingly parallel: can be split into sub-tasks run simultaneously (e.g., sim with varying parameters)
- inherently serial: cannot be split into concurrent sub-tasks (e.g., `run_sim_robust_se.R` with one replicate per job)

Job arrays make embarrassingly parallel tasks easy:

- unique identifier for an entire set of jobs (easy to manage)
- unique identifier for each task within array (set simulation parameters)
- displays nicely on cluster (part of not flooding)

Using the cluster: batch submission scripts

Your turn!

Check out the file `submit_sim_robust_se.sh`. **With a partner**, answer the following questions:

1. What do lines 3 and 4 do?
2. How many jobs are in my array if I want 5000 total jobs and 50 replicates per job?

Using the cluster: batch submission

Your turn!

Run the executable file `submit_sim_robust_se.sh`, with command line arguments

- `"robust_se"`
- 5000
- 50

Using the cluster: `simulator`

`simulator` on the cluster: you have to grab nodes manually.

Using the cluster: `simulator`

`simulator` on the cluster: you have to grab nodes manually.

Then, pass them as a character list into the argument `parallel`, e.g.,

Using the cluster: `simulator`

`simulator` on the cluster: you have to grab nodes manually.

Then, pass them as a character list into the argument `parallel`, e.g.,

```
list_of_names <- <get correct names>
simulate_from_model(nsim = 1000,
       index = 1:3,
       parallel = list(socket_names = list_of_names))
```

Using the cluster: checking and altering jobs

Many options once you have submitted a job:

Using the cluster: checking and altering jobs

Many options once you have submitted a job:

- `qstat` checks queue status;

Using the cluster: checking and altering jobs

Many options once you have submitted a job:

- `qstat` checks queue status; helpful args:
  - `f` shows full status of jobs
  - `u <user>` shows status only for `user` (e.g., `brianw26`)
  - `j <jobnum>` shows details for a single job

Using the cluster: checking and altering jobs

Many options once you have submitted a job:

- `qstat` checks queue status; helpful args:
  - `f` shows full status of jobs
  - `u <user>` shows status only for `user` (e.g., `brianw26`)
  - `j <jobnum>` shows details for a single job
- `qhold <job_id>[.tasklist]` puts a hold on job `job_id` (and optionally array elements in `tasklist`)

Using the cluster: checking and altering jobs

Many options once you have submitted a job:

- `qstat` checks queue status; helpful args:
  - `f` shows full status of jobs
  - `u <user>` shows status only for `user` (e.g., `brianw26`)
  - `j <jobnum>` shows details for a single job
- `qhold <job_id>[.tasklist]` puts a hold on job `job_id` (and optionally array elements in `tasklist`)
- `qrls <job_id>` removes a hold on a job
- `qdel <job_id>` deletes a job
- `qalter <job_id>` alters a job

Using the cluster: other helpful SGE commands

Remember: `bayes` is a shared resource!

Using the cluster: other helpful SGE commands

Remember: `bayes` is a shared resource!

Rule of thumb: 20 jobs running on cluster at once.

Using the cluster: other helpful SGE commands

Remember: `bayes` is a shared resource!

Rule of thumb: 20 jobs running on cluster at once.

Ways to help share the cluster:

Using the cluster: other helpful SGE commands

Remember: `bayes` is a shared resource!

Rule of thumb: 20 jobs running on cluster at once.

Ways to help share the cluster:

- Submit batch jobs in job arrays

Using the cluster: other helpful SGE commands

Remember: `bayes` is a shared resource!

Rule of thumb: 20 jobs running on cluster at once.

Ways to help share the cluster:

- Submit batch jobs in job arrays
- Use holds on jobs, using `tc` argument in `qsub`

Using the cluster: other helpful SGE commands

Remember: `bayes` is a shared resource!

Rule of thumb: 20 jobs running on cluster at once.

Ways to help share the cluster:

- Submit batch jobs in job arrays
- Use holds on jobs, using `tc` argument in `qsub`
- Estimate timing of smallest job, consider this when creating and submitting job arrays

Using the cluster: checking and altering jobs

Your turn!

1. Check the status of **your** job array (replace my netid with yours): `qstat -f -u brianw26`
2. Allow only 20 jobs to run at a time (replace `<job_id>` with your job id): `qalter <job_id> -tc 20`

Using the cluster: wrapping up

After all jobs are finished, pull results back to your machine.

Using the cluster: wrapping up

After all jobs are finished, pull results back to your machine.

Then run your downstream code to compile results!

Using the cluster: wrapping up

After all jobs are finished, pull results back to your machine.

Then run your downstream code to compile results!

Your turn!

1. Pull results files back to your computer
2. Run `load_sim_robust_se.R`
3. What can you conclude based on these results?

Using the cluster: wrapping up

Today, you practiced how to

Using the cluster: wrapping up

Today, you practiced how to

- run code without a GUI,

Using the cluster: wrapping up

Today, you practiced how to

- run code without a GUI,
- use shell scripts,

Using the cluster: wrapping up

Today, you practiced how to

- run code without a GUI,
- use shell scripts,
- build a simulation,

Using the cluster: wrapping up

Today, you practiced how to

- run code without a GUI,
- use shell scripts,
- build a simulation,
- code modularly,

Using the cluster: wrapping up

Today, you practiced how to

- run code without a GUI,
- use shell scripts,
- build a simulation,
- code modularly,
- and use the department cluster!

Using the cluster: wrapping up

Today, you practiced how to

- run code without a GUI,
- use shell scripts,
- build a simulation,
- code modularly,
- and use the department cluster!

These skills are easily portable to other cluster systems and your own coding projects (e.g., `R` packages).

Using the cluster: wrapping up

Today, you practiced how to

- run code without a GUI,
- use shell scripts,
- build a simulation,
- code modularly,
- and use the department cluster!

These skills are easily portable to other cluster systems and your own coding projects (e.g., `R` packages).

Remember to be nice: the cluster is a shared resource!

Appendix: other useful cluster things

Useful things: Windows file compatability

Editing files in Windows carries risks.

One such risk is adding *control characters* that cannot be processed on Unix systems (e.g., Linux on department cluster).

A helpful tool to remove this characters is `dos2unix`: to remove control characters from `myfile.sh`, run `dos2unix myfile.sh`.

Useful things: emails and other defaults

You can set default behavior (on `bayes`, but similar for other systems) by creating a file called `.sge_request`.

Mine reads

```
-j y
-cwd
-S /bin/bash
-q normal.q
-M brianw26@uw.edu
-m e
```

Which means:

- submit either binary or script file
- run using `bash`
- email me
- email me at end of job

Useful things: aliases and functions (Mac/Linux)

You will probably end up logging into the cluster or sending files back and forth quite often.

Having *aliases* and *bash functions* set up makes this easier.

Creating aliases:

1. go to your `.ssh` folder (in your home directory on your computer)
2. create a file called `config` (using, e.g., vim)
3. edit it using the template below

```
Host <replace with, e.g., bayes>
    HostName <replace with, e.g., bayes.biostat.washington.edu>
    User <replace with, e.g., brianw26>
```

Useful things: Useful things: aliases and functions (Mac/Linux)

Creating functions:

1. go to your home directory
2. create a file called, e.g., .bash_funcs
3. edit it using the template below
4. edit your .bashrc file to include the lines
   ```
   if [ -f ~/.bash_funcs ]; then
       . ~/.bash_funcs
   fi
   ```
   (makes sure that .bash_funcs is sourced when you open a new Terminal window)

```
function_name () {
    commands
}
```

E.g. : send R files from cwd to specified directory on bayes:

```
send_r_bayes () {
    scp *.R brianw26@bayes.biostat.washington.edu:~/$1
}
```

Useful things: FileZilla (all, but esp. Windows)

FileZilla is a nice program for pulling results back to your computer (via `scp`).

It can be used on all types of computers, but is especially helpful for Windows (since command prompt is strange!).