R-BASED HPC AT MIT FOR POLITICAL SCIENCE

Weihuang Wong / March 23, 2015

OBJECTIVES

- Why HPC?
- The *engaging1* cluster
- Connecting to *engaging1* and initial setup
- Parallelization in `R`
- The `Slurm` scheduler

WHY PARALLELIZE?

When application is *embarrassingly parallel*, parallelization yields large performance boost with minimal programming overhead, e.g.

- Fitting same model on different datasets
- Resampling methods e.g. bootstrap
- Monte Carlo algorithms, e.g. JAGS

HPC cluster allows us to parallelize across dozens or hundreds of processors.

THE *ENGAGING1* CLUSTER

- Collaborative project run by several universities in the Boston area
- Used by several research groups at MIT e.g. EAPS, Nuclear Science lab, etc.
- 234 compute nodes
- Each node: two 8-core (=16 cores) 2GHz Intel Xeon E2650 processors, 64GB of memory

CONNECTING TO *ENGAGING1*

Connect using `ssh`:

```
$ ssh student001_17806@eofe4.mit.edu
```

Download setup scripts from Git repo:

```
[student001_17806@eofe4 ~]$ git clone https://github.com/ins
```

SETUP

```
[student001_17806@eofe4 ~]$ cd eofe-scripts/
[student001_17806@eofe4 ~]$ source setup.sh
```

The setup script makes R and other applications available at startup, downloads RMPI, and installs RMPI and other packages for parallelization.

PARALLELIZATION IN R

Packages make parallelization easy, e.g. *foreach* and *snow*

Here's a toy example with *foreach*:

```r
library(doParallel)

sayhello <- function() {
    info <- Sys.info()[c("nodename", "machine")]
    Sys.sleep(runif(1, 0, 50)/10)
    paste("Hello from", info[1], "with CPU type", info[2], "at", Sys.time())
}

cl <- makeCluster(24, type="MPI")   # Init MPI backend
registerDoParallel(cl)
hello.out <- foreach(i=1:24, .combine='c') %dopar% sayhello()
stopCluster(cl)                     # Clean up
print(hello.out)
mpi.quit()
```

We are initializing a cluster with 24 workers (a.k.a. processors/cores or sometimes also "slaves").

THE `SLURM` SCHEDULER

It is good practice **not** to run programs interactively in a HPC environment.

Use only computing resources allocated to us explicitly by the scheduler.

We use a *Slurm batch script* to request for resources.

SLURM SCRIPT TEMPLATE

```
#!/usr/bin/env bash
#SBATCH -p sched_17806          # name of runtime queue
#SBATCH --nodes=2               # number of nodes
#SBATCH --ntasks-per-node=13    # number of cores
#SBATCH -t 1:00                 # wall-time (mm:ss)
#SBATCH -J barebones            # job name
#SBATCH -o log.%j               # output

echo '=========='
cd ${SLURM_SUBMIT_DIR}
echo ${SLURM_SUBMIT_DIR}
echo Running on host $(hostname)
echo Time is $(date)
echo SLURM_NODES are $(echo ${SLURM_NODELIST})
echo '=========='
mpirun -n 1 Rscript --no-save --no-restore --verbose barebones.r > bb.Rout 2>&1
```

Why are we asking for 2x13 workers when we initialized a cluster with 24 workers in R?

Within R, maximum number of workers we can request is total number of workers requested in Slurm

WORKING WITH `SLURM`

- Send batch script to Slurm: `sbatch <filename>`

```
[student001_17806@eofe4 demo]$ sbatch run_bb.slurm
Submitted batch job 815349
```

- Monitor job progress: `squeue -u <username>` or `sacct -u <username>`
- Can also give `squeue` and `sacct` job ID: `-j <jobid>`
- To see the entire queue: `qstat`
- Cancel a job: `scancel -j <jobid>`

JOB OUTPUT

Let's look at our .Rout file:

```
[wwong@eofe4 barebones]$ more bb.Rout
running
  '/cm/shared/engaging/R/3.1.1/lib64/R/bin/R --slave --no-restore --no-save --nc

Loading required package: foreach
Loading required package: iterators
Loading required package: parallel
Loading required package: Rmpi
        24 slaves are spawned successfully. 0 failed.
[1] 1
 [1] "Hello from node107 with CPU type x86_64 at 2015-03-23 15:00:21"
 [2] "Hello from node107 with CPU type x86_64 at 2015-03-23 15:00:20"
...
[22] "Hello from node108 with CPU type x86_64 at 2015-03-23 15:00:19"
[23] "Hello from node108 with CPU type x86_64 at 2015-03-23 15:00:20"
[24] "Hello from node108 with CPU type x86_64 at 2015-03-23 15:00:21"
```

CONCLUSION

Parallelization on *engaging1* is easy! The return on investment is high.

- If you know how to parallelize on local machine or Athena, you're almost there.
- `man <command>` is a useful resource
- More details in tutorial PDF