STAT DSCP: Data Science Computing Project

Homework 3: Distributed Computing via Slurm and the Statistics High Performance Computing (HPC) Cluster

1. Login to a suitable HPC computer.

   - Login to `slurm-submit-00.cs.wisc.edu`. Use `slurm-submit-00` only for editing and running Slurm commands that launch and manage jobs. (Do not run computations here, as it cannot handle computations from many people.)
   - Run `cd /workspace/STATuser` (using your STATuser name) to work in a directory the compute nodes can read. (They cannot read your home directory.)

   (Optional: From a second terminal, run `srun --pty /bin/bash` to get an interactive job on a compute node where you can run and debug computations from a terminal. I do this from within the emacs shell (in the second terminal). You may ignore the message "`bash: .../.bashrc: Permission denied`" which occurs because the compute nodes cannot read your home directory.)

2. Solve the `mtcars` exercise at
   `www.stat.wisc.edu/~jgillett/DSCP/HPC/examples/5mtcarsPractice/instructions.txt`.

   Hint: I recommend that you now go to step (4) and turn in an incomplete but working version of your work. (We will grade your last submission before the deadline.)

   Since this exercise (2) started as group work, it is ok for your solution to look like the solution of members of your group. For exercise (3), below, you should do independent work, so your solution should not look like other students' solutions.

3. Read

   `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/HG7NV7`,

   which links to and describes data on all U.S. flights in the period 1987-2008. Find out, for departures from Madison:

   - How far you can get in one flight?
   - What is the average departure delay for each day of the week?

   To do this, write a program `submit.sh` and supporting scripts to:

   (a) Use `sbatch --partition short` to run 22 parallel jobs, one for each year from 1987 to 2008. The first job should:

      i. download the 1987 data via
      `wget http://pages.stat.wisc.edu/~jgillett/DSCP/HPC/airlines/1987.csv.bz2`

      ii. unzip the 1987 data via `bzip2 -d 1987.csv.bz2`

iii. use a short bash pipeline to extract from `1987.csv` the columns `DayOfWeek` (1=Monday through 7=Sunday), `DepDelay`, `Origin`, `Dest`, and `Distance`; and retain only the rows whose `Origin` is `MSN` (Madison's airport code); and write a much smaller file, `MSN1987.csv`.

The other 21 jobs should handle the other years analogously.

(b) When those parallel jobs are done, use `sbatch --partition short` to accomplish these tasks:

- Collect the Madison data from your 22 `MSN*.csv` files into a single `allMSN.csv` file.
- Answer "How far can you get from Madison in one flight?" Write a line like `MSN,ORD,109` to answer. This line says, "You can fly 109 miles from Madison (MSN) to Chicago (ORD)." But 109 isn't the farthest you can get from Madison in one flight; write the correct line. (Hint: I used a `bash` pipeline to do this.) Save the result in `farthest.txt`.
- Answer "What is the average departure delay for each day of the week?" Write a pair of lines like these to a file `delays.txt`:

  ```
  Mo  Tu  We  Th  Fr  Sa  Su
  8.3 5.0 4.3 5.5 9.5 2.1 3.5
  ```

  (These are not the correct numbers.)
  Hint: I used R's `tapply()` to do this. e.g. Here is an example of using `tapply()` to find the average gas mileage for each number of cylinders in a car's engine:

  ```
  > tapply(X=mtcars$mpg, INDEX=mtcars$cyl, FUN=mean)
         4        6        8
  26.66364 19.74286 15.10000
  ```

- To save disk space, delete the large data files `1987.csv` through `2008.csv`.
  Hint: You might want to comment out the download, extract, and delete commands while developing your code so you don't have to wait for downloading every time you run your jobs.

4. Organize files to turn in your solution. (See "Copying files with `scp`," below.)

(a) Make a directory `NetID_hw3`, where `NetID` is your NetID.

(b) Make a subdirectory `NetID_hw3/mtcars`. Copy only these files there:

- `getData.sh`
- `jobArray.sh`
- `findLightest.sh`
- `submit.sh`
- `out`

We should be able to recreate `out` by running `./submit.sh`.

(c) Make a subdirectory `NetID_hw3/airlines`. Copy only these files there:

- `submit.sh`
- `farthest.txt`
- `delays.txt`

- any supporting *code* files required by your `submit.sh` (do not submit data files)

We should be able to recreate `farthest.txt` and `delays.txt` by running `./submit.sh`.

(d) Make a file `README` in the directory `NetID_hw3` with a line of the form `NetID,LastName,FirstName`. If you collaborated with any other students on the mtcars part of this homework, add additional lines of this form, one for each of your collaborators. So, for example, if George Box with NetID `gepbox` worked with John Bardeen with NetID `jbardeen`, George's `README` file should look like

```
gepbox,Box,George
jbardeen,Bardeen,John
```

(e) From the parent directory of `NetID_hw3`, run `tar cvf NetID_hw3.tar NetID_hw3` and then upload `NetID_hw3.tar` as your HW3 submission on Canvas.

To verify your submission, download it from Canvas, and then:

i. Make a directory to test in, e.g. `mkdir test_HW3`.

ii. Move your downloaded `.tar` file there and extract it.

iii. In each of the `NetID_hw3/mtcars` and `NetID_hw3/airlines` directories, run `./submit.sh` and check that the correct output is produced (and that the large files `1987.csv` through `2008.csv` are gone).

Copying files with `scp`

`scp` ("secure copy") copies files between a local machine and a remote machine. (`rsync` is another commonly-used program for this task.) Our linux handout mentions this form:
`scp [[user@]host:]file1 [[user@]host2:]file2]`, where the things in square brackets `[]` are optional. For example, from your laptop:

- copy `file.txt` from `slurm-submit-00.cs.wisc.edu` to your local machine via
  `scp STATuser@slurm-submit-00.cs.wisc.edu:/workspace/STATuser/file.txt file.txt`
- copy `file.txt` from your local machine to `slurm-submit-00.cs.wisc.edu` via
  `scp file.txt STATuser@slurm-submit-00.cs.wisc.edu:/workspace/STATuser/file.txt`

See `man scp` for more information.

Caution

In a recent semester, about half of the students turned in solutions that did not do parallel computing. Use `sbatch ... --array=...` to get parallel computing. On a recent run, my 22 jobs took from 18 to 154 seconds to run, with an average of about 111 seconds. This would cost about (22 jobs)(111 seconds/job)(1 minute / 60 seconds) = 40.7 minutes if run serially. It only takes about 2 minutes when run in parallel.