Stat243: Problem Set 6, Due Friday November 8

October 29, 2019

This covers Units 7 and 8.

- It's due at 2 pm on Friday November 8, **both submitted as a PDF to bCourses as well as committed to your Github repository**.
- Please note my comments in the syllabus about when to ask for help and about working together. In particular, **please give the names of any other students that you worked with on the problem set and indicate in comments any ideas or code you borrowed from another student.**
- The formatting requirements are the same as previous problem sets, but given that problems 1 and 2 ask you to run code on a remote machine, you won't execute those chunks in your document. Just show your code and the plotting and timing results.
- When you use srun or sbatch to submit a job to the SCF cluster you MUST use the *–cpus-per-task* flag to specify the number of cores that your computation will use. In your Python or R code you can then either hard-code that same number of cores as the number of workers or (better) you can use the SLURM_CPUS_PER_TASK UNIX environment variable to tell Dask or future how many workers to start. We should have discussed this in section but I think we forgot to do so.

Problems

1. With the full Wikipedia traffic data for October-December 2008 (available on Savio in */var/local/s243/wikistats_full/dated* on any of the low partition SCF cluster nodes), explore the variation over time in the number of visits to Barack Obama-related Wikipedia sites, based on searching for "Barack_Obama". You should use Dask to do the filtering and grouping by day-hour. You can do this either in an interactive session using *srun* or a batch job using *sbatch*. And if you use *srun*, you can run Python itself either interactively or as a background job. Once you have done the filtering and gotten the counts for each day-hour, you can simply use standard R or Python code on your laptop to do some plotting to show how the traffic varied over the days of the full time period and particularly over the hours of November 3-5, 2008 (election day was November 4 and Obama's victory was declared at 11 pm Eastern time on November 4).

   Notes:

   (a) Note that I'm not expecting you to know any more Python than we covered in Section, so feel free to ask for help (and for those of you who know Python to help out) on Python syntax on the bCourses discussion board or in office hours.

   (b) There are various ways to do this using Dask bags or Dask data frames, but I think the easiest in terms of using code that you've seen in Unit 8 is to read the data into Python and do the

filtering using a Dask bag and then convert the Dask bag to a Dask dataframe to do the grouping and summarization (the grouping/summarization is also illustrated in the Unit 8 code). The file *ps6.R* has some additional code hints. Alternatively you should be able to use *foldby()* from *dask.bag*, but figuring out what arguments to pass to *foldby()* is a bit involved.

(c) Make sure to test your code on a portion of the data before doing computation on the full dataset. **Reading and filtering the whole dataset will take something like 60 minutes with 16 cores. You MUST test on a small number of files on your laptop or on one of the stand-alone SCF machines (e.g., radagast, gandalf, beren) before trying to run the code on the full 120 GB (zipped) of data.**

(d) Please do not use more than 16 cores in your SLURM job submissions so that cores are available for your classmates. If your job is stuck in the queue you may want to run it with 8 rather than 16 cores.

2. This question asks you to complete the exercise begun in section on October 15. Consider the Wikipedia traffic data as in Problem 1.

(a) Using the future package, with either *future_lapply* or *foreach* with the *doFuture* backend, write code that, in parallel, reads in the data and filters to only the rows that refer to pages where "Barack_Obama" appears. You can use the code from *unit7-parallel.R* as a template, in particular the chunks labeled 'rf-example', 'foreach' and 'future_lapply'. Collect all the results into a single data frame. You can do this either in an interactive session using *srun* or a batch job using *sbatch*. And if you use *srun*, you can run R itself either interactively or as a background job.
IMPORTANT: for this, use only the 64 files in */var/local/s243/wikistats_full/dated_for_R*. Why? For some reason each R worker builds up to using about 10 GB of memory when reading the files in *dated* (despite gc() reporting only 5 GB used per worker), which would be 160 GB of memory across 16 workers. I don't want to have multiple users on the same node in a situation where the node runs out of memory. The files in dated_for_R are smaller than those in dated (and are only the first 64 of about 1000 files in total)
Hints: (a) *readr::read_delim()* should be quite fast if employed effectively, (b) there are lines with fewer than 6 fields, but *read_delim()* should still work and simply issue a warning, and (c) there are lines that have quotes that should be treated as part of the text of the fields and not as separators. Also, as above, you should test your code on a small subset interactively first.

(b) Compare the timing of selecting the Obama-related subset with Dask versus R's future package using the same number of cores. The 64 files in *dated_for_R* are in total only 1/16 of the full dataset, so make the necessary arithmetic adjustment when comparing future to Dask. (Also note that because the nodes scf-sm00,...,scf-sm03 are older and slower than scf-sm10,...,scf-sm13, this may not be a completely apples-to-apples comparison if you end up on a different node when running the R job than the Python job.

3. Using the Stack Overflow database (http://www.stat.berkeley.edu/share/paciorek/stackoverflow-2016.db), write SQL code that will determine which users have asked only R-related questions and no Python-related questions. Those of you with more experience with SQL might do this in a single query, but it's perfectly fine to create one or more views and then use those views to get the result as a subsequent query. Report how many unique such users there are. The Stack Overflow SQLite database is ~ 650 MB on disk, which should be manageable on most of your laptops, but if you run into problems, you can use an SCF machine.