Problem Set 5

Due Monday Oct. 28, 10 am

Comments

- This covers material in Units 6 and 7.
- It's due at 10 am (Pacific) on October 28, both submitted as a PDF to Gradescope as well as committed to your GitHub repository.
- Please see PS1 for formatting and attribution requirements.
- Running parallel code, working with large datasets, and using shared computers can result in delays and unexpected problems. Don't wait until the last minute. The lengthy time period to complete this problem set is simply a logistical result of when the quiz is scheduled – you should start on the problem set before the quiz. We are happy to help troubleshoot problems you may have with accessing the SCF, running parallel Python code, submitting jobs to the SCF cluster, etc. We will not be happy to do so if it is clear that you started at the last minute.
- Given that you'll be running batch jobs and operating on remote servers, for problems 1 and 2 you'll need to provide your code in chunks with `#| eval: false`. You can paste in any output you need to demonstrate your work. Remember that you can use " "‘" to delineate blocks of text you want printed verbatim.
- To monitor an `sbatch` job, see this SCF doc.

Problems

1. This problem asks you to use Dask to process some Wikipedia traffic data and makes use of tools discussed in Section on October 11. The files in `/scratch/users/paciorek/wikistats/dated_2017_small/dated` (on the SCF) contain data on the number of visits to different Wikipedia pages on November 4, 2008 (which was the date of the US election in 2008 in which Barack Obama was elected). The columns are: date, time, language, webpage, number of hits, and page size. (Note that the Unit 7 Dask bag example and Question 2 below use a larger set of the same data.)

    a. In an interactive session on the SCF Linux cluster (ideally on the `low` partition, but if necessary on the `high` partition), start an interactive session on one of the cluster nodes using `srun` (as discussed in section). Request four cores. Then follow these steps:

        i. Copy the data files to a subdirectory of the `/tmp` directory of the machine your interactive session is running on. (Keep your code files in your home directory.) Putting the files on the local hard drive of the machine you are computing on reduces the amount of copying data across the network (in the situation where you read the data into your program multiple times) and should speed things up in step ii.

ii. Write efficient Python code to do the following: Using the `dask` package as seen in Unit 6, with either `map` or a list of delayed tasks, write code that, in parallel, reads in the space-delimited files and filters to only the rows that refer to pages where "Barack_Obama" appears in the page title (column 4). You can use the code from Unit 6 as a template. Collect all the results into a single data frame. In your `srun` invocation and in your code, please use four cores in your parallelization so that other cores are saved for use by other users/students. IMPORTANT: before running the code on the full set of data, please test your code on a small subset first (and test your function on a single input file serially).

iii. Tabulate the number of hits for each hour of the day and make a (time-series) plot showing how the number of visits varied over the day (I don't care how you do this - using either Python or R is fine.). Note that the time zone is UTC/GMT, so you won't actually see the evening times when Obama's victory was announced - we'll see that in Question 2. Feel free to do this step outside the context of the parallelization. You'll want to use datetime functions from Pandas or the `datetime` package to manipulate the timing information (i.e., don't use string manipulations).

iv. Remove the files from `/tmp`.

Tips:

1. In general, keep in mind various ideas from Unit 2 about reading data from files. A couple things that posed problems when I was prototyping this in using `pandas.read_csv` were that there are lines with fewer than six fields and that there are lines that have quotes that should be treated as part of the text of the fields and not as separators. To get things to work ok, I needed to set the `dtype` to `str` for the first two fields (for ease of dealing with the date/time info later) but NOT set the `dtype` for the other fields, and to use the `quoting` argument to handle the literal quotes.
2. When starting your `srun` session, please include the flag `--mem-per-cpu=5G` when submitting the Slurm job. In general one doesn't need to request memory when submitting jobs to the SCF cluster but there is a weird interaction between Dask and Slurm that I don't quite understand that requires this.

b. Now replicate steps (i) and (ii) but using `sbatch` to submit your job as a batch job to the SCF Linux cluster, where step (ii) involves running Python from the command line (e.g., `python your_file.py`. You don't need to make the plot again. As discussed here in the Dask documentation, put your Python/Dask code inside an `if __name__ == '__main__'` block.

Note that you need to copy the files to `/tmp` in your submission script, so that the files are copied to `/tmp` on whichever node of the SCF cluster your job gets run on. Make sure that as part of your `sbatch` script you remove the files in `/tmp` at the end of the script. (Why? In general `/tmp` is cleaned out when a machine is rebooted, but this might take a while to happen and many of you will be copying files to the same hard drive so otherwise `/tmp` could run out of space.)

2. Consider the Wikipedia traffic data for October 15-November 15, 2008 (already available in `/var/local/s243/wikistats/dated_2017_sub` on all of the SCF cluster nodes in the low or high partitions). As in Question 1, explore the variation over time in the number of visits to Barack Obama-related Wikipedia sites, based on searching for "Barack_Obama" on English language

Wikipedia pages. You should use Dask with distributed data structures to do the reading and filtering, as seen in Unit 7. Then group by day-hour (it's fine to do the grouping/counting in Python in a way that doesn't use Dask data structures). 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. Time how long it takes to do the Dask part of the computations to get a sense for how much time is involved working with this much data. Once you have done the filtering and gotten the counts for each day-hour, you can simply use standard Python or R code on your laptop to do some plotting to show how the traffic varied over the days of the full month-long 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:

- 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 7 is to read the data in and do the filtering using a Dask bag and then convert the Dask bag to a Dask dataframe to do the grouping and summarization. Alternatively you should be able to use `foldby()` from `dask.bag`, but figuring out what arguments to pass to `foldby()` is a bit involved.
- 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 30 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, arwen) before trying to run the code on the full 40 GB (zipped) of data. For testing, the files are also available in `/scratch/users/paciorek/wikistats/dated_2017_sub`.
- When doing the full computation via your Slurm job submission:
  - Don't copy the data (unlike in Question 1) to avoid overloading our disks with each student having their own copy. Just use the data from `/var/local/s243/wikistats/dated_2017_sub`.
  - 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.
  - As discussed in Section, when you use `sbatch` to submit a job to the SCF cluster or `srun` to run interactively, you should be using the `--ntasks-per-node` flag (`--cpus-per-task` is fine too) to specify the number of cores that your computation will use. In your Python code, you can then either hard-code that same number of cores as the number of workers or (better) you can use the `SLURM_NTASKS` shell environment variable to tell Dask how many workers to start (or `SLURM_CPUS_PER_TASK` if using `--cpus-per-task`).

3. SQL practice using the Stack Overflow database.

   a. Find all the users (including their `displayname`) who have asked but never answered a question. You're welcome to set up your solution using views, but you should not need to. In your solution, provide as many separate queries as needed to illustrate **all** of the following SQL functionality: a subquery in the WHERE statement, a subquery in the FROM statement, a set operation using INTERSECT, UNION or EXCEPT), and an outer join (note that SQLite does not have a right outer join). Check the count of the number of results from the different queries to make sure they are the same.

Note that for reasons I haven't looked into, both the questions and answers tables contain rows where the `ownerid` is `NULL`. These can mess up certain versions of the queries I'm asking you to create. So first create views that omit those cases from those two tables. (One could do the omitting within the main query, but this is a good situation for using views to make the main query easier to understand.)

b. Do the different approaches vary much in terms of how much time they take? What about using SQLite versus DuckDB?