Stat243: Problem Set 6, Due Monday Nov. 2

October 18, 2015

Comments:

- This covers Units 7 and 8.
- It's due at the start of class on Nov. 2.
- As usual, your solution should mix textual description of your solution, code, and example output.
- Please note my comments in the syllabus about when to ask for help and about working together.
- Please give the names of any other students that you worked with on the problem set.
- Please see *howtos/startEC2virtualMachine.txt* and *howtos/startSparkVirtualCluster.txt* for instructions on starting up a single Amazon EC2 instance (virtual machine) and an Amazon EC2 Spark virtual cluster, respectively. We've tested these out, but I wouldn't be surprised if there are issues, so don't spin your wheels much before checking with us (in office hours or via Piazza) about problems.
- The airline dataset is available at *http://www.stat.berkeley.edu/share/paciorek/* as a set of .bz2 files (*1987.csv.bz2*, ... , *2008.csv.bz2*) and a tarball containing all of the .bz2 files that may be more convenient to download (*1987-2008.csvs.tgz*).
- Some of your computations in both SQL and Spark may take a while (e.g., 5-60 minutes). You may want to experiment with a smaller subset of the data while you get your code working.
- **Don't delay in getting started on this. A lot of what you'll try out here is likely new to many of you, and things are more complicated logistically to do in the cloud than just doing something on your local machine. You'll probably have to look up some SQL and Python syntax. We're happy to provide hints and to help as you run into snags, but not, for example, on the Sunday before it's due.**

Problems

1. Using an Amazon EC2 instance with at least 4 cores (you'll use the 4 cores in Problem 3), create an SQLite database from R with a single table to hold the airline data. To do this you should not read the entire dataset into memory in R, even if you have a machine with sufficient memory. Rather, use the individual year files and build up your table in pieces. Note that because RSQLite appears to convert NAs to zeros in numeric fields, you should replace the missing values in the fields used in Problem 2 with a numeric code.
   How big is the database file, compared to the original CSV (12 Gb) and a bzipped copy of the original CSV (1.7 Gb)?

2. Create a Spark RDD containing the airline data. Use 12 slave nodes in your virtual cluster. Load the individual .bz2 files onto the HDFS and read the data into a Spark RDD. Repartition so the dataset is equitably spread across the worker nodes in your Spark cluster. Now try the following:

   (a) Subset/filter your datasets in your Spark and SQLite representations to omit flights with missing values for the departure delay or values that seem unreasonable. In the remaining questions, use this filtered dataset.

   (b) Now do some exploration where you compare the speed with Spark and SQLite of aggregating information into categories where there is one category for each unique combination of airline, departure airport, arrival airport, calendar month (January through December), day of week (Monday through Sunday), and hour of day of the scheduled departure time. You should extract the following summary statistics for each category (i.e., key): proportion of flights more than 30 minutes late, proportion more than 60 minutes late, and proportion more than 180 minutes late.

   Note: in Spark, you should be able to use *reduceByKey()* rather than *groupByKey()*. When you do so, if your reduction function is trying to do vectorized addition, you should make sure it is operating on a *numpy* array, not a regular Python array. Also, for Spark, feel free to compute the counts needed to compute the proportions without actually finishing the calculation of computing the proportions.

   (c) For Spark, get the resulting aggregated dataset onto the disk of the master node as a single data file. (Note that to use *saveAsTextFile()*, you likely need to do an additional map step such that each observation is a comma-delimited character string of information.)

   (d) Add an index to your SQLite database. How does having an index affect the speed of your query?

   (e) Using your results from either SQLite or Spark, report the top 5 or 10 groupings (keys) in terms of proportion of late flights for groupings with at least 150 flights.

   (f) Extra credit: Experiment with some variations on the Spark calculations. Some things you could consider are as follows. Look at the information in the section on "RDD Persistence" in the Spark Programming manual. Experiment with using the *cache()* method when carrying out multiple operations in separate lines of code. Does it seem to make a difference? How about forcing the RDD to be stored on disk rather than in memory? Does changing the number of partitions affect timing? How about changing the number of reduce tasks in your *reduceByKey* operation? One of the Spark developers indicated to me that using only as many reduce tasks as cores in the cluster would be a reasonable strategy.

   Note: when you use Spark, it uses lazy evaluation, so sometimes you may have to do an operation like `take(1)` to ensure that the computation actually gets done. So the timing numbers may be a bit fuzzy.

3. Use *foreach*, *mclapply()*, or one of the parallel apply functions in R on your EC2 instance. How does the speed of calculating the proportion of delayed flights by key in parallel using the SQLite database compare to the other approaches used above? Note that you'll need to open separate database connections for the separate tasks for this to operate in parallel. Make sure you do this on the table with the index from Problem 2c already created or this may take a really long time, probably because the multiple R instances querying the single database interfere with each other. (In fact, how this works when multiple queries are made is something we would want to look further into if we were doing this as part of a real workflow.) Also you'll need to think about how many tasks to parallelize over.

4. Use bash tools to cut out the columns not needed. Don't explicitly unzip the files before the cutting - rather use piping to directly produce output .bz2 files from the input .bz2 files. How long does this take on an m3.xlarge instance? Does it seem like a worthwhile preprocessing step?