PDAT 611 Module 6 Assignment: Spark

Scott Thatcher

4/19/2021

Introduction

The purpose of this part of the assignment is to explore the data set size at which Spark becomes faster than pure R code. As mentioned in the lectures, Spark has a certain amount of overhead—communication between nodes across the network, etc. For relatively small tasks, it can be quicker to simply run code in R. Spark's performance is tied to many choices in configuration settings and to the nature of the problem itself, so any absolute numbers you see in this assignment shouldn't be assumed to be universally true for any data set or problem. But, it is good to get a sense for the trade-offs you run into between the two approaches as your data set gets larger.

As mentioned in the lecture, there are several data sets containing U.S. airline flight data stored both in the `/opt/Data` directory on fire and in the `/user/spark/Data` directory in the distributed HDFS file system. In this part, you'll write code to do two different computational tasks, and test that code against the flights data sets of varying sizes.

Although I've tried to make tasks that are at least believably of interest, this assignment is more concerned with exploring the process than in producing any great insight about flight times from the work you're doing.

For each step outlined below, use the `system.time` function to measure the time it takes that step to be completed, and record your times in a table like the one shown below:

Data Set	`2009.csv`	`flights_09-10.csv`	`flights_09-11.csv`	…
num. rows	6429338	…	…	…
R load time	16 sec.	…	…	…
Spark load time	6 sec.	…	…	…
summarize/filter w/ R	10 sec.	…	…	…
summarize/filter w/ Spark	10 sec.	…	…	…
lin. model w/ R	11 sec.	…	…	…
lin. model w/ Spark	11 sec.	…	…	…

Loading Data

First you'll want to load the data in two different ways:

- Load the data normally into R, for example using `read.csv()`.
- Load the data into Spark with `spark_read_csv()`.

When you load the data, it will be useful (in terms of this experiment) to select only the variables that you will

need, and omit rows with NA values. So, your data loading commands might look something like this:

```
system.time({
flts <- read.csv("/opt/Data/2009.csv") %>%
  select(OP_CARRIER, ORIGIN, DEST, ARR_DELAY, DEP_DELAY, DISTANCE) %>% ... %>%
  na.omit()
})
system.time({
flts_tbs <- spark_read_csv(sc, name="flts_tbs",
  path = "hdfs://fire.truman.edu:9000/user/spark/Data/2009.csv") %>%
  select(OP_CARRIER, ORIGIN, DEST, ARR_DELAY, DEP_DELAY, DISTANCE) %>% ... %>%
  na.omit()
})
```

where you may want to put more code where you see `...` in order to make sure the variables are assigned the right type.

Calculating Average Arrival Delay

Next, write a string of R code to do the following:

1. Within a `system.time` command, create a single piped set of expressions to transform the `flts` data frame, according to the steps below.

- Calculate the mean arrival delay for each carrier and each origin/destination pair. In other words, you should be calculating one average delay for American Airlines flying from JFK to LAX, one average delay for American Airlines flying from JFK to SAN, etc.
- Then in the same pipe, filter the results to include only a single origin and destination of your choice. This should yield a table with one or more airlines all with the same origin and destination. Do this after the step above.
- Finally, store the results in a new R data frame.

2. Within a new system.time command, Create code that mirrors what you did in Step 1 above, but this time transforming the Spark data table `flts_tbl` and storing it in a new Spark table.
3. Outside of the timing, display this new table, perhaps using `kable`.

Calculating a Linear Model

Next write code that does the following:

1. Outside of timing, creates a new data frame and a new Spark table by filtering your `flts` and `flts_tbl` to pick out only flights that originated in New York's JFK airports.
2. With these new data frames, create linear models using both R and Spark to predict `ARR_DELAY` as a function of `DEP_DELAY`, `DISTANCE`, `OP_CARRIER` and `DEST`. Time both these blocks of code.
3. Outside of timing, use both R and Spark to predict the arrival delay of a flight from JFK to MIA whose distance is 1090 miles, whose carrier is American Airlines (AA), and whose departure delay is 0.

Write a Final Summary

Show your table of timings, and write a paragraph that summarizes any patterns you saw in the timings for the various tasks you did with the variously-sized data files.

What to Turn In

- Turn in a RMarkdown file along with a knit PDF document that shows your results from each section.
- Since you'll be doing several runs with different data files, the knit document can simply be the result of one of those runs, but your final table of values in the knit document will reflect data you collected from all of your runs.

Reminders and Notes

- You don't have to try every single data file, but you should try a single year, the biggest multi-year set, and at least one in between.
- Variable names are the same in all data files (but different from those in the nycflights13 data set). So, once you write the code once, you should be able to simply change the file names and run it again on a different data set.
- Note that I haven't asked you to print a summary of the linear models because that output would get long.
- To get the consistent time estimates, run each timed block more than once—this makes sure the data is most likely to be cached in memory, so that you're really comparing the algorithms more than one-off data-load times. **Don't do that with read.csv and spark_read_csv commands, however!**
- Since I've asked you to run the timed code blocks more than once to let the times settle in, your final table with timings should be produced "by hand," showing representative times that you recorded from these runs. You *shouldn't* construct your table by automatically recording the times you got during the final knitting process.