Exercise 10

Due Friday July 23 2021.

Some files are provided that you need below: E10.zip. As usual, **you may not write any loops** and **all calculations must be done with Spark DataFrames**.

Reddit Average Scores

Another data set we have seen before: the Reddit comment archive *(http://files.pushshift.io/reddit/)* . Like the GHCN weather data, when we saw it before, I had extracted summary data that you needed for the question. The full data set is still too much to expect you to work with, but we can work with some reasonable subsets (and it is on the cluster if you actually need it for some reason).

Let's ask the question: **what is the average score in each subreddit**? It would be interesting to know if some communities are "friendlier" than others. We have all of the comments with their scores, so getting an average shouldn't be too hard.

Create a program `reddit_averages.py` that takes input and output directories on the command line. The provided hint gives a schema for the Reddit data (more on that below).

We want to produce the average scores sorted two ways: by subreddit name, and by score (highest scores first). We will put these in two separate output directories (as the hint implies).

Spark Overhead

Let's have a look at how things are running in your `reddit_averages.py`, and see some places we can be more clever in what we tell Spark about our calculations.

For this part, I suggest you work locally (not the cluster: it will be easier to control and see the timing that way) with the `reddit-2` input data set. It's on the CSIL Linux computers, or downloadable as described in the data sets instructions.

Even with this much data, in order to actually see any human-perceivable differences, we'll need to slow things down. Add the `--master=local[1]` switch to your command line: this tells Spark to run single-threaded. Also add `time` to measure the time the job takes. So your command should be like:

```
time spark-submit --master=local[1] reddit_averages.py reddit-2 output
```

Have a look at the "real" time taken (that's wall-clock time *(https://en.wikipedia.org/wiki/Wall-clock_time)*).

You'll be reporting your results here in the `answers.txt` below.

Startup

Start by timing how long `reddit_averages.py` takes to run on the two-line data set, `reddit-0`. This will give us a baseline of how long it takes **any** local Spark job to start up and shut down, even if it's not really doing any work.

Side lesson: any non-Spark program you have that takes less time than this cannot possibly be sped up by Spark. [On the cluster, the same thing takes about 30 seconds: the magic YARN does isn't free either.]

The Schema

A quirk of the Spark JSON reading: if you don't specify a schema, it reads the entire dataset to determine it. Then it reads the whole thing **again** to create the DataFrame.

Prove this to yourself: time the program with and without the `schema=` argument when reading the input (likely the `reddit-2` data set).

Replace the `schema=` argument for the next part…

DataFrame Re-Use

In your `reddit_averages.py`, you likely have some pattern more-or-less like this:

```
averages = comments.…
averages.sort(…).write.csv(out + '-subreddit')
averages.sort(…).write.csv(out + '-score')
```

This uses the `averages` DataFrame twice. Because of the lazy evaluation of DataFrames, what actually happens there might surprise you: the JSON file is read, the averages are calculated, sorted by subreddit, the result is written, and the **result is discarded**. Then the whole thing needs to start again (from reading the input) to produce the by-score output.

If you want to use a DataFrame's contents again, you have to tell Spark to *cache* it. Modify your program so the averages are only calculated once. Something like this:

```
averages = averages.cache()
```

Time it again: with and without the `.cache()`.

When you submit your `reddit_averages.py`, give the fastest version.

Popular Wikipedia Pages

Wikipedia publishes (used to publish?) page view statistics *(https://dumps.wikimedia.org/other/pagecounts-raw/)* . These are summaries of page views on an hourly basis. The file (named `pagecounts-20160801-120000.gz` for a particular hour) contains lines like this:

```
en Aaaah 20 231818
en Aaadonta 2 24149
en AaagHiAag 1 8979
```

That means that during this hour (Aug 1, 2016 from 12:00–13:00), the English Wikipedia page ("en") titled "Aaaah" was requested 20 times, returning a total of 231818 bytes.

For this question, create a Spark job that **finds the the most-viewed page each hour** and how many times it was viewed. The program should be called `wikipedia_popular.py` and (as usual) take input and output directories from the command line.

To refine the question a little, we're interested in: (1) English Wikipedia pages (i.e. language is `"en"`) only. (2) The most frequent page is usually the front page (title is `'Main_Page'`) but that's boring, so **exclude it** from the results. (3) Also, "special" (titles starting with `'Special:'`) are boring and should **also be excluded**. [The smaller data sets with subsets of pages might not have the main page or special pages: that doesn't mean you aren't responsible for this behaviour.]

You can assume the filenames will be in the format `pagecounts-YYYYMMDD-HHMMSS*`. We want the `YYYYMMDD-HH` substring of that as a label for the day/hour.

Some hints:

- The DataFrame.read.csv *(https://spark.apache.org/docs/3.1.2/api/python/reference/api/pyspark.sql.DataFrameReader.csv.html)* function can read these space-delimited files with a `sep` argument. It can also have meaningful column names if you give a `schema` argument.
- The filename isn't obviously-visible, but can be retrieved with the `input_file_name` function if you use it right away: `spark.read.csv(…).withColumn('filename', functions.input_file_name())`
- Unless you're more clever than me, converting an arbitrarily-deep filename (that might be `file:///` or `hdfs:///` and `.gz` or `.lz4` or neither, or who-knows what else) is very hard in the DataFrame functions. I suggest writing a Python function that takes pathnames and returns string like we need: `'20160801-12'`. Then turn it into a UDF: `path_to_hour = functions.udf(…, returnType=types.StringType())`.
- To find the most-frequently-accessed page, you'll first need to find the largest number of page views in each hour.
- … then join that back to the collection of all page counts, so you keep only those with the `count == max(count)` for that hour. (That means if there's a tie you'll keep both, which is reasonable.)

Sort your results by date/hour (and page name if there's a tie) and output as a CSV. Part of my output on `pagecounts-1` looks like this: (note the tie in hour 6)

```
20160801-05,Simon_Pegg,109
20160801-06,Simon_Cowell,96
20160801-06,Simon_Pegg,96
20160801-07,Simon_Pegg,101
20160801-08,Simon_Pegg,119
20160801-09,Simon_Cowell,127
```

Questions

Answer these questions in a file `answers.txt`.

1. How long did your `reddit_averages.py` take with (1) the `reddit-0` data set and effectively no work, (2) no schema specified and not caching (on `reddit-2` for this and the rest), (3) with a schema but not caching, (4) with both a schema and caching the twice-used DataFrame? [The `reddit-0` test is effectively measuring the Spark startup time, so we can see how long it takes to do the actual work on `reddit-2` in the best/worst cases.]
2. Based on the above, does it look like most of the time taken to process the `reddit-2` data set is in reading the files, or calculating the averages?
3. Where did you use `.cache()` in your `wikipedia_popular.py`? [Hint: the answer had better be "once"… but where?]

Submitting

Submit your files through CourSys for Exercise 10.

Updated Mon July 12 2021, 15:12 by ggbaker.