Exercise 9

Due Friday November 13 2020.

Some files are provided that you need below: E9.zip. As usual, **you may not write any loops** in your code.

When asked for Spark programs, **all calculations must be done with Spark DataFrames**. With small test data sets, it may be possible to bring the data into a Pandas DataFrame (or similar) and do the calculation locally. Of course, that's not the point: keep the data in Spark until we have some rules for when it can be safely collected back to the driver process.

Aside: Big Data data files

Distributing input data for this part of the course is challenging. I will distribute a few different input sets so you can test your programs at different scales. See some more detail on the data sets instructions page.

tl;dr: You'll get a toy-sized data set in the `.zip` file you're used to downloading. Bigger data sets can be found to do more serious "big data" testing.

For the exercises (and in general with big data), you should start with small data sets to get the basics working, then scale up as you work out performance problems.

Getting Started with Spark

For this question, see the program `first_spark.py` in this week's `.zip`. We want to get it running on your computer (or a CSIL Linux workstation): see the Running Spark Jobs Locally instructions for guidance. This is generally the right way to start working on Spark programs: locally with small data sets where you can test quickly.

This program takes the `xyz-*` data sets as input. These are randomly-generated values to give us something to work with. The program groups the values by their `id % 10` value, sums the `x` values, and outputs in CSV files. [Don't look for any meaning in these calculations: they are a random thing to calculate so we can see Spark do something.]

You should be able to run it with a command like:

```
spark-submit first_spark.py xyz-1 output
```

Assuming it runs correctly, you should be able to see its output (in CSV files) with a command like this:

```
cat output/part-* | less
```

To make sure you have to at least look at the code, **add a column to the output** which is the mean average of the `y` values in each group. (Hint: avg *(https://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions.avg)* .)

Getting Started with our cluster

Now let's get the program working on our Cluster. Assuming your code worked locally, it *should* work on the cluster.

See the cluster instructions for more detailed information, but you need to SSH to `gateway.sfucloud.ca` and always start your session with the command:

```
module load 353
```

Copy your program (by SCP/SFTP) to `gateway.sfucloud.ca` and you should be able to submit the job to the cluster:

```
spark-submit first_spark.py /courses/353/xyz-2 output
```

The output files will be stored on the cluster's HDFS. That means an extra step to view them. See the HDFS instructions for more details, but since the output is uncompressed and quite small, you can simply:

```
hdfs dfs -cat output/*
```

YARN Web Frontend

Running this job on the `xyz-3` data set should take a measurable amount of time, even on the cluster. While it's running, we can explore the information YARN (the thing responsible for doing compute work on the cluster) gives us about jobs. Visit http://localhost:8088 *(http://localhost:8088)* (assuming you have done the port forwarding as described in the Cluster instructions. If not and you're on-campus, you can probably access at http://master.sfucloud.ca:8088 *(http://master.sfucloud.ca:8088)*).

You should see a list of jobs that are running or have been run on the cluster. Find yours (maybe by selecting "RUNNING" while your job is still active and find it in that much-shorter list) and make a note of its "application ID" in the first column of the table: you need to **give it as an answer below**. [Once you have that, feel free to the following tour of the frontend in small groups: there's not much utility to each of you hammering away at the cluster to each see the same screen.]

Click the application ID and the "ApplicationMaster" link. Have a look around and see the information given about your job and what's happening. The frontend will disappear when the task completes: either run it again if you want to explore more, or decide you're done your tour. (The YARN frontend in general is always there: just not for your job after it completes.)

If you feel like it, you can also explore the HDFS web frontend at http://localhost:50070 *(http://localhost:50070)* or http://master.sfucloud.ca:50070 *(http://master.sfucloud.ca:50070)* .

Extract-Transform-Load GHCN data

We have worked several times with data from the GHCN *(https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/global-historical-climatology-network-ghcn)* . I have always extracted the parts you needed: the full data set is >160MB of gzipped CSV per year. It is cumbersome to deal with without big data tools. But now…

When doing work with the maximum daily temperatures, it's not completely trivial to get that out of the original data files. This is very much an ETL task, but one where the big data tools become necessary.

Write a program `weather_etl.py` that takes two command-line arguments: the input and output directories, so it can be started like this:

```
spark-submit weather_etl.py weather-1 output # locally
spark-submit weather_etl.py /courses/353/weather-1 output # cluster
```

You'll find some code to get you started in `weather_etl_hint.py`. There is a directory of data files distributed as `weather-1` in the ZIP file. You'll also find (as described above) medium- and large-sized data sets `weather-2` and `weather-3`. These are partitioned subsets of the original GHCN data, but the exact same file format.

Things that need to be done to get the data into a more usable shape:

1. Read the input directory of `.csv.gz` files.
2. Keep only the records we care about:
    a. field `qflag` (quality flag) is null;
    (Hint *(https://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.Column.isNull)*)
    b. the `station` starts with `'CA'`; (Hint option 1 *(https://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.Column.startswith)* ; Hint option 2 *(https://spark.apache.org/docs/3.0.0/api/python/pyspark.sql.html#pyspark.sql.functions.substring)*)
    c. the `observation` is `'TMAX'`.
3. Divide the temperature by 10 so it's actually in °C, and call the resulting column `tmax`.
4. Keep only the columns `station`, `date`, and `tmax`.
5. Write the result as a directory of JSON files GZIP compressed (in the Spark one-JSON-object-per-line way).

Here are two lines of correct output (after uncompressing with `gunzip` or similar)

```
{"station":"CA001173242","date":"20161203","tmax":1.0}
{"station":"CA004038116","date":"20161203","tmax":4.0}
```

Since the output here is compressed and a little larger, you'll need to uncompress and page the output to view it:

```
cat output/* | zless # local
hdfs dfs -cat output/* | zless # cluster
```

Reddit Average Scores

I had originally put the “Reddit Average Scores” question on this exercise, but moved it to next week. You might want to start looking at that question and “Spark Overhead” from Exercise10 this week, but I won't force you to submit (until next week).

Questions

Answer this question in a file `answers.txt`.

1. What was the application ID you saw in the YARN frontend (http://localhost:8088 *(http://localhost:8088)*) of the job where you actually ran the `first_spark.py` on the cluster? (It should be like `application_0000000000000_0000`. If you rand multiple time, pick any one.)

Submitting

Submit your files through CourSys for Exercise 9.

Updated Wed Sept. 09 2020, 16:53 by ggbaker.