CS 348 - Project 3

SQL on Spark!

Step 2: Executing SQL queries on Apache Spark.

Due on: Check the Blackboard or Course schedule or Piazza.

Note: There will be a 10% penalty for each late calendar day. After five calendar days, the project will not be accepted.

Overview

This project will show you how to execute SQL queries on Apache Spark.

Environment

NOTE: **Purdue Career Account User ID** will be represented here as `$USER` .We will be using the same cluster specified in step 1 handout.

Spark provides APIs for Scala, Java, and Python. For this project we will use Python.

For an introduction to Spark, take a look at the following links.

http://spark.apache.org/docs/2.4.3/quick-start.html
http://spark.apache.org/docs/2.4.3/rdd-programming-guide.html
http://spark.apache.org/docs/2.4.3/sql-programming-guide.html
https://www.slideshare.net/LisaHua/spark-overview-37479609

NOTE: Be mindful of the version number whenever you are searching online for documentation. The version of Spark installed on the CS department's Spark cluster is version 2.4.3. Spark is under heavy development, so if you are looking at documentation for another version, there is a good chance that features will be changed or missing.

Log in to `jacobi00` (This is the master node). Spark provides a Python shell where you can submit commands interactively. This shell can be very useful when trying to learn the Spark interface. To start the shell, run:

```
$ pyspark
```

If you like, you can develop your queries in the shell first, and then copy your shell commands to a Python script to create a Python job. To distribute a Python job with Spark, use:

```
$ spark-submit --master yarn [job.py] [command_line_args]
```

Dataset

Spark can read data from a number of sources, include a local file system, HDFS, HBASE, etc. For this project, we will load our data into HDFS and then read the data from HDFS into Spark.

We will use exactly the same source datasets as in setp 1. If you have already loaded the `movies_utf8.dat`, `users_utf8.dat`, and `ratings_utf8.dat` datasets into your HDFS directory, you do not need to reload them.

As directed in step 1, make sure you are using the MovieLens 1M dataset (not the 100K dataset) and that you have cleaned the data before loading it into HDFS.

Example

We will provide you with the code (`q1.py`) to complete one query:
/* Query 1 (example query) */
/* Give the distinct title of each movie that received the rating 1 at least once. */

```
SELECT DISTINCT m.title
FROM
movies AS m
JOIN ratings AS r ON m.movieid = r.movieid
WHERE r.rating = 1;
```

To run the query, copy the file `q1.py` to your home directory on the master node and run the following command:

```
$ spark-submit --master yarn q1.py [input_dir] [output_dir]
```

(i.e.)

```
$ spark-submit --master yarn q1.py /user/$USER/in /user/$USER/out
```

NOTE: It takes 30-45 seconds to run and produce results.

CAUTION: Hadoop will complain if you try to write to an HDFS directory that already exists. Either you change the output directory name to create a new one or remove the current output directory.

Next, check the output:

```
$ hdfs dfs -ls /user/$USER/out
```

You should see an output like this with `SUCCESS` message:

```
Found 11 items
-rw-r--r--   1 $USER supergroup          0 2019-10-15 17:49 /user/manig/out/_SUCCESS
-rw-r--r--   1 $USER supergroup       9397 2019-10-15 17:49 /user/manig/out/part-00000
-rw-r--r--   1 $USER supergroup       9601 2019-10-15 17:49 /user/manig/out/part-00001
-rw-r--r--   1 $USER supergroup       8611 2019-10-15 17:49 /user/manig/out/part-00002
-rw-r--r--   1 $USER supergroup       7700 2019-10-15 17:49 /user/manig/out/part-00003
-rw-r--r--   1 $USER supergroup       8050 2019-10-15 17:49 /user/manig/out/part-00004
-rw-r--r--   1 $USER supergroup       8715 2019-10-15 17:49 /user/manig/out/part-00005
-rw-r--r--   1 $USER supergroup       8661 2019-10-15 17:49 /user/manig/out/part-00006
-rw-r--r--   1 $USER supergroup       6384 2019-10-15 17:49 /user/manig/out/part-00007
-rw-r--r--   1 $USER supergroup       7072 2019-10-15 17:49 /user/manig/out/part-00008
-rw-r--r--   1 $USER supergroup       8011 2019-10-15 17:49 /user/manig/out/part-00009
```

NOTE: If you look in `q1.py`, we use `repartition()` function to specify the number of partitions considering the number of cores and the amount of data you have. That is why you see 10 parts (0-9) in the given output.

To see the contents of the output, run

```
$ hdfs dfs -cat /user/$USER/out/*
```

Sometimes it may not display anything. In that case, use the following:

```
$ hdfs dfs -cat '/user/$USER/out/*'
```

Move the output from HDFS back to your directory on the master node:

```
$ hdfs dfs -getmerge /user/$USER/out ./q1_output.txt
$ cat q1_output.txt
```

Finally, remove the output directory so that you can run your job again. (Hadoop will complain if you try to write to an HDFS directory that already exists.)

```
$ hdfs dfs -rm -r /user/$USER/out
```

(`-rm -r` is used for removing directories in HDFS.)

Tasks

We ask that you implement the following queries as Spark jobs. For each query, we demonstrate what a single line of output from your final job should look like.

1. (25 points) /* Query 2 */
   Give the distinct title and genre of each movie released in 2000 that received the rating 5 at least once.

   /* output */
   Almost Famous (2000)::Comedy—Drama

2. (25 points) /* Query 3 */
   For each occupation, compute the number of users with that occupation.

   /* output */
   18::70

3. (25 points) /* Query 4 */
   Give the title and average rating for each movie released in 1998. Round the average to the nearest tenth.

   /* output */
   After Life (1998)::4.1

4. (25 points) /* Query 5 */
   For each user age group, compute the average rating given by the users in that age group. Round the average to the nearest tenth.

   /* output */
   25::3.5

Important
Because the Spark SQL library is well developed, some of these queries can be completed using a call to `sqlContext.sql()`. For other queries, you may have to do some post-query processing.

When writing your queries in Spark, take care that you do not use any Spark SQL keywords as column names. This will give you an error.

NOTE: Queries 4 and 5 ask you to compute averages. Depending on how you compute your average, your solution may differ from our solution by ±0.1. There will not be any penalty for this.

Deliverables

Note that it is not sufficient to simply execute your queries in the Spark shell and turn in your query results. You must collect your commands in a Python job. We should be able to launch your python job using:

```
$ spark-submit --master yarn [job.py] [path_to_input_dir] [output_dir]
```

To get full credit on the assignment, you must carefully follow the instructions below. Make sure to name your files as indicated. You should turn in your code and results file for Query 1 (the example query) and for Queries 2-5. (You may turn in the Query 1 code without modification.)

Create a directory with your Purdue User ID as the directory name. (Here, by Purdue ID we mean your career account username.) In this directory, create five subdirectories, one for each query. Within each query directory, place the code for your Python job as well as a single file with your query results (retrieved from HDFS with `-getmerge`).

```
[$USER]/
            q1/
               q1.py
               q1_results.txt
            q2/
               q2.py
               q2_results.txt
            ...
```

Navigate to the parent directory of your [`$USER`] directory and run this command:

```
$ zip -r [$USER].zip [$USER]
```

Submit the file `[$USER].zip` to Blackboard.