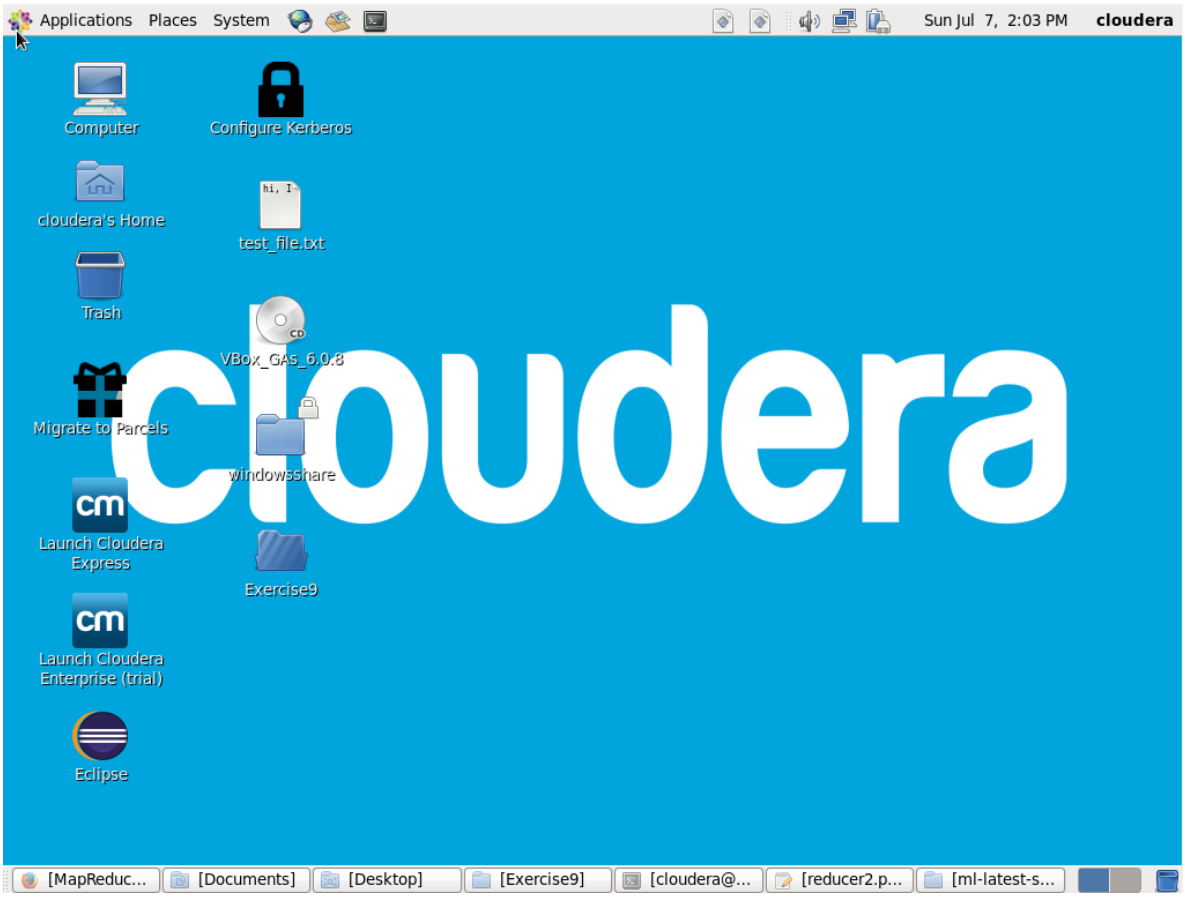
Distributed Data Analytics Lab

Sebastian Pineda Arango Mtr. Nr. : 246098

Exercise 9

In this laboratory, we are going to present some exercises don in Hadoop, using Map-Reduce (with Hadoop streaming) and Python as programming language. Everything was done in a Cloudera-Quickstart virtual machine [1], since in this way the configuration difficulties can be avoided, meanwhile we leverage all the Cloudera pre-installed features. In the following image, we can see how the deskop of the installed virtual machine looks like:



This virtual machine has the following characteristics:

- Red hat operative system (64 bits)
- 4 GB RAM
- 1 Core

We are aware that though this configuration may be useful for academic purposes, it is not suitable completely to do performance analysis.

It is also important to point out that for the whole development of this lab, the resource found in [2] has been very valuable, since it is explained there how to use Python with Hadoop streaming.

Analysis of Airport Efficiency with Map Reduce

1. **Compute the maximum, minimm and average for every airport**

For this task we use a file which has the following fields: (FL_DATE, OP_CARRIER_AIRLINE_ID, OP_CARRIER_FL_NUM, ORIGIN, DEST, DEP_TIME, DEP_DELAY, ARR_TIME, ARR_DELAY) and which is used as input for the mapper.

- **Mapper (mapper1.py):** for each input line, the mapper outputs a line with the following format: (ORIGIN, DEP_DELAY, 1). It means that in this case, ORIGIN is the key.
- **Reducer (reducer1.py):** the reducer has to compute the minimum, the maximum and the average of the departure delay (DEP_DELAY) for every airport (ORIGIN). To compute the average, it aggregates the delays and the counts. Once there is a new airport (ORIGIN value), it restarts the count. Of course, this can be done since there is a shuffle instance before the reducer that ensures that the keys are ordered. During the reducing step, the minimum and maximum departure time per airport is also kept and compared successively. The reducer only has to compare with the previous DEP_DELAY of the same key (ORIGINA) values to make this.

The following command is used to execute the Map-Reduce jobs (using Hadoop-streaming):

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -mapper "python $PWD/mapper1.py" -reducer "python $PWD/reducer1.py" -input "transportation_statistics.csv" -output "out_1"
```

The content of the output file can be accessed through the following command:

```
hdfs dfs -cat  out_1/part-00000
```

The output of the reducer looks like this:

```
ID:"ABE"        Mean:20.49      Min:-11.0       Max:794.0
ID:"ABI"        Mean:25.79      Min:-11.0       Max:263.0
ID:"ABQ"        Mean:8.55       Min:-18.0       Max:911.0
ID:"ABR"        Mean:36.24      Min:-13.0       Max:1259.0
ID:"ABY"        Mean:10.01      Min:-21.0       Max:291.0
ID:"ACT"        Mean:12.95      Min:-17.0       Max:202.0
ID:"ACV"        Mean:10.16      Min:-17.0       Max:578.0
ID:"ACY"        Mean:7.71       Min:-21.0       Max:670.0
ID:"ADK"        Mean:0.67       Min:-34.0       Max:41.0
ID:"ADQ"        Mean:-6.07      Min:-28.0       Max:73.0
ID:"AEX"        Mean:11.55      Min:-20.0       Max:354.0
ID:"AGS"        Mean:20.97      Min:-11.0       Max:1130.0
ID:"ALB"        Mean:11.75      Min:-21.0       Max:981.0
ID:"AMA"        Mean:4.42       Min:-15.0       Max:298.0
ID:"ANC"        Mean:3.62       Min:-48.0       Max:1195.0
ID:"APN"        Mean:19.04      Min:-19.0       Max:278.0
ID:"ASE"        Mean:31.64      Min:-20.0       Max:1110.0
ID:"ATL"        Mean:15.07      Min:-22.0       Max:1470.0
ID:"ATW"        Mean:24.85      Min:-16.0       Max:1065.0
ID:"AUS"        Mean:8.72       Min:-22.0       Max:1246.0
ID:"AVL"        Mean:14.63      Min:-14.0       Max:425.0
ID:"AVP"        Mean:32.04      Min:-18.0       Max:783.0
ID:"AZO"        Mean:9.33       Min:-20.0       Max:268.0
ID:"BDL"        Mean:6.65       Min:-20.0       Max:549.0
ID:"BET"        Mean:-2.48      Min:-27.0       Max:72.0
ID:"BFL"        Mean:-2.52      Min:-20.0       Max:290.0
ID:"BGM"        Mean:9.02       Min:-15.0       Max:216.0
ID:"BHM"        Mean:13.75      Min:-18.0       Max:1068.0
ID:"BIL"        Mean:7.03       Min:-19.0       Max:340.0
ID:"BIS"        Mean:10.41      Min:-18.0       Max:189.0
```

2. Compute a ranking list that contains top 10 airports by their average arrival delay

In this exercise, we use the same input as the las exercise for the mapper.-

- **Mapper (mapper2.py):** Per line it outputs a tuple (DEST, ARR_DELAY, 1), therefore it discards the minimum and maximum value.
- **Reducer (reducer2.py):** Takes the output of the mapper, accumulates the delays and the counts, computes the averages and, finally, aggregates by ordering. Then it returns the top 10 airports with highest mean delay.

The execution of the mapper and reducer is similar to the previous exercise. The output of the reducer looks as follows:

```
{'"ELP"': 81.77, '"BQK"': 47.86, '"GJT"': 46.38, '"BNA"': 37.58, '"ABQ"':
34.14, '"LAX"': 29.73, '"MAF"': 29.06, '"GRI"': 24.98, '"CHO"': 24.64,
'"ACV"': 24.04, '"OAK"': 23.69}
```

Analysis of movie dataset using Map and Reduce

For this exercise, we use the small Movielens-Dataset for the sake of ease and speed in computation. For all the tasks we use the following command to change the number of mappers and reducers:

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -D
mapred.map.tasks=2 -D mapred.reduce.tasks=1 -mapper "python
$PWD/mapper.py" -reducer "python $PWD/reducer.py"   -input "ml-latest-
small/ratings.csv"  -output "out_4"
```

In the following, we do not plot the results, but we show them in tables, since it is easier to appreciate how it change. However, we can observe that the time increases as we increase the number of mappers a reducers. This is probably because we are running in a virtual machine, which has very low limited resources and has only one configured PC.

1. **Find the movie title which has the maximum average rating.**

To calculate the maximum average rating we create a mapper (**mapper3.py**) which outputs a tiplet of (movieId, rating, 1):

>> movieId1 rating1 1

>> movieId2 rating2 1

Then, the reducer (**reducer3.py**), aggregates the ratings (accumulates) and the 1, so that we have a count of every movie. Then, both results are used to calculate the mean (average). At the same time, the reducer keeps track of the maximum value. Since there could be several, it will only return only one movie.

After executing the following command:

```
[cloudera@quickstart Exercise9]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-str
eaming.jar -mapper "python $PWD/mapper3.py" -reducer "python $PWD/reducer3.py"
 -input "ml-latest-small/ratings.csv"  -output "out_4"
```

We get the list of different movies ID with the mean, and finally, one of the movies with the highest average rating.

```
ID:99813        Mean:3.88
ID:99846        Mean:3.88
ID:99853        Mean:3.88
ID:999  Mean:3.04
ID:99910        Mean:3.25
ID:99917        Mean:3.17
ID:99992        Mean:3.17
The movie with max. average rating has ID 115111 which is 5.0
```

Number of mappers	Number of reducers	Execution time
1	1	1 min 1 sec.
2	1	1 min 18 sec
2	2	1 min 26 sec
3	1	3 min 12 sec

2. **Find the user who has assigned the lowest average rating among all the users who rated more than 40 times.**

The input for the taks is the file "rating.csv".

- **Mapper (file mapper4.py):** It takes as input a file, where every line has the structure: (*userId, movieId, rating, timestamp*). It outputs a line for every register transformed to the following structure (*userId, rating, 1*).
- **Reducer one (file reducer4.py)**: It takes the output of the first mapper and accumulates the ratings and the counts for every user, and calculates the mean (ratings sum divided by the total count). Then, it aggregates the results by finding iteratively the minimal average, but only does so if the total count is greater than 40.

Finally we get the following output:

```
The user ID 139 has the lowest rating mean (2.14)
```

Number of mappers	Number of reducers	Execution time
1	1	43 sec.
2	1	2 min. 15 sec.
2	2	1 min. 54 sec
3	1	2 min. 2 sec.

3. **Find the highest average rated movie genre**

For this task, the input is a file which is the join between “ratings.csv” and “movies.csv”. In this way, we have the rating and the genres for every movie in a single file. Then, we use two map and reduce jobs which perform as follows.

- **Mapper (file mapper5.py):** It takes lines from the merged file which have the following structure: (userId, movieId, rating, timestamp, genres) and output a line for every genre contained in the line “genres” (these are separated by pipe |). Every output line of the mapper has the structure (genre, rating, 1). So for example, if the input line is:

  >> (100, 10, 3.5, 1111, Action|Comedy)

  The output would be:

  >> (Action, 3.5, 1)
  >>(Comedy, 3.5, 1)

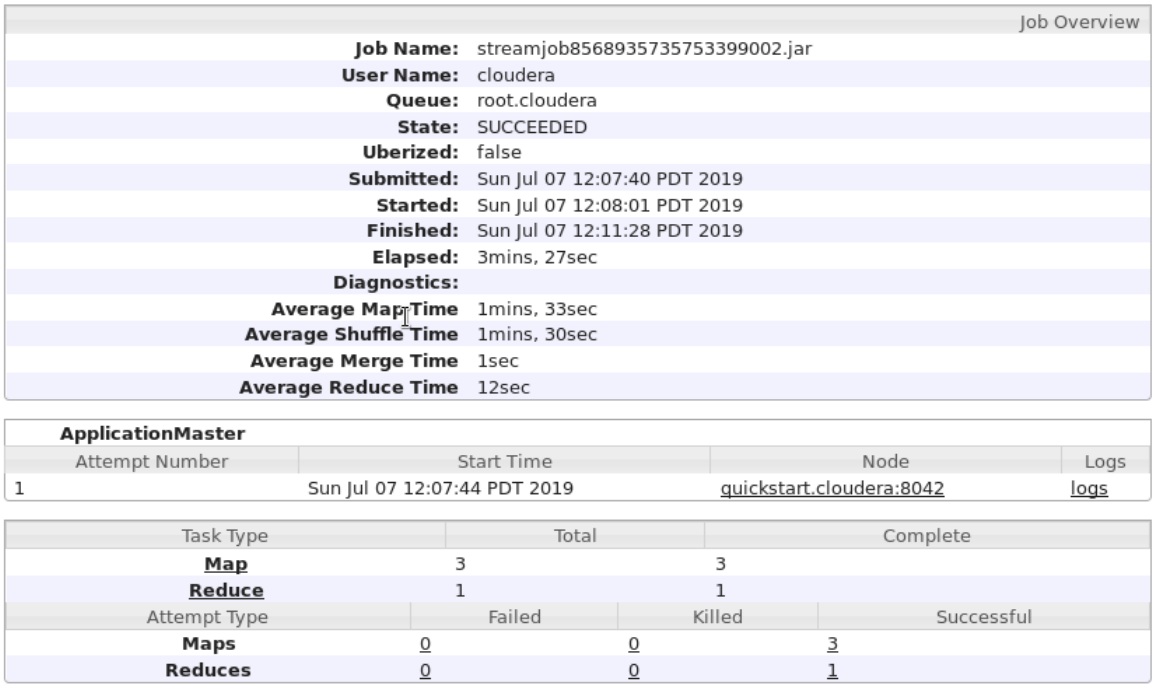
- **Reducer (file reducer5.py):** The reducer accumulates the rating and the count, in order to compute the mean rating. It, moreover, aggregates this information by aggregating the maximal average value among them.

Number of mappers	Number of reducers	Execution time
1	1	58 sec.
2	1	1 min. 29 sec.
2	2	1 min. 46 sec.
3	1	3 mins 27. Sec.

Finally, after running on Hadoop, we get the following output:

```
Genre:Horror    Mean:3.62
```

Here, we can see a screenshot of the Haddop interface used to measure the time:



Job Overview	
Job Name:	streamjob8568935735753399002.jar
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Sun Jul 07 12:07:40 PDT 2019
Started:	Sun Jul 07 12:08:01 PDT 2019
Finished:	Sun Jul 07 12:11:28 PDT 2019
Elapsed:	3mins, 27sec
Diagnostics:	
Average Map Time	1mins, 33sec
Average Shuffle Time	1mins, 30sec
Average Merge Time	1sec
Average Reduce Time	12sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun Jul 07 12:07:44 PDT 2019	quickstart.cloudera:8042	logs

Task Type	Total	Complete
Map	3	3
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	3
Reduces	0	0	1

References:

[1] Virtual Image: https://www.cloudera.com/downloads/quickstart_vms/5-13.html

[2] Map-Reduce with Python tutorial: https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/