Cloud Computing Project

Capstone – Spark Project

Erwan Fouillen

1. Introduction

The goal of this second task on spark is to showcase the simultaneous application of several big data technologies, e.g.:

- Data loading
- In-memory distributed computing
- Parallelized SQL (in our case, with Hive and/or Cassandra)
- Containerized applications (use of Docker to run Spark and Cassandra together)

Please note that, as I had issues with AWS credits, I ended up doing everything locally with Docker:

1. Use of a bash script to download data files concurrently (see docker image in resources)
2. Data loading with Spark DataFrames
3. Query the data with SQL statements (Hive) or CQL statements (Cassandra
4. Store the results in a Cassandra database (in my case, PrestoDB)
5. GitHub repository: https://github.com/ErwFoui/CloudComputingProject

2. Data Extraction

As I didn't use AWS to complete this assignment, I directly downloaded the data from the website of the US Bureau of Transportation Statistics. Plainly put, and as shown in the **get_data.sh** file of my github repository, I parallelized in a bash script the download of monthly flight CSVs from 1987 to 2008 as follows:

1. **cURL** https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236 with a SQL statement specifying the required fields plus the **year** and **month** filters. The required fields include those useful to answer the assignment's questions, i.e.: YEAR, MONTH, DAY_OF_MONTH, DAY_OF_WEEK, FL_DATE, OP_UNIQUE_CARRIER, OP_CARRIER_FL_NUM, ORIGIN_AIRPORT_ID, ORIGIN, DEST_AIRPORT_ID, DEST, CRS_DEP_TIME, DEP_TIME, DEP_DELAY, DEP_DELAY_NEW, CRS_ARR_TIME, ARR_TIME, ARR_DELAY, ARR_DELAY_NEW, CANCELLED
2. The server answers by specifying the location of the zipped file for the given year/month filters. **wget** is used to download the file at the URL sent by the server.
3. Zipped files are unzipped and renamed.
4. The 21 * 12 csv files are downloaded in parallel by forking bash processes (see **get_data.sh)**:

```
max_num_processes=$(ulimit -u)
limiting_factor=4
num_processes=$((max_num_processes/limiting_factor))

# Download all CSVs in parallel
for year in `seq 1987 2008`
  do
  for month in `seq 1 12`
    do
    ((i=i%num_processes)); ((i++==0)) && wait
    download_csv $year $month &
    done
  done
```

5. Once download is complete, data is extracted from the CSV files with Spark's DataFrame API, in a SparkContext or a SQLContext.
6. As running Kafka locally proved too slow when sending data to/consuming data from the Kafka, I ended up using directly Spark to extract and concatenate the CSV files in lazily evaluated DataFrames.

3. Architecture

All in all, the architecture is very similar to that offered by AWS, i.e., use of PySpark to distribute files across Resilient Distributed Datasets, then use Hive-like statements to query these files with simple SQL queries, and finally store tables in a database (Cassandra). Everything was done locally with the help of the following Docker image: https://github.com/Yannael/kafka-sparkstreaming-cassandra.

All a user has to do is to start this docker image with a volume linking the host machine's /Data folder to the container's directories, run **get_data.sh** to concurrently download all the files, and finally run any of the 3 Jupyter notebooks to extract and process the data. Please note that, as I was only using my personal laptop, RAM capabilities prevented me from keeping all the CSV files in memory. Consequently, I ended up using datasets from 1995 (instead of 1987) to 2008, hence some discrepancies between results from part 1 and those from part 2.

4. Results

This last part shows answers to the assignment's questions. All queries and results are shown in the **SparkQueries – Q1/2/3** files. Please note that, as there is no missing data in the csv files downloaded at https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236 (contrary to some parts of the AWS snapshot), actual results might slightly change. However, the relative rankings match those indicated on the Capstone project's webpage.

1.1

ORIGIN	**all_flights**
ORD	9252980
ATL	8885867
DFW	7926248
LAX	5844795
PHX	5047005
DEN	4508601
IAH	4423809
DTW	4197592
LAS	4024215
MSP	3939329

1.2

airline	**avg_arr_delay**
HA	-0.775
KH	1.157
F9	5.693
WN	5.84
OO	5.877
9E	6.108
TZ	6.129
NW	6.298
US	6.471
DH	6.798

1.3

```
+---+-------------+
|dow|avg_arr_delay|
+---+-------------+
|  6|        4.281|
|  2|        6.008|
|  3|        7.172|
|  7|        7.249|
|  1|        7.297|
|  4|        9.341|
|  5|       10.257|
+---+-------------+
```

2.1

```
+------+-------+---------+----+
|origin|carrier|avg_delay|rank|
+------+-------+---------+----+
|   CMH|     FL|    1.948|   1|
|   CMH|     DH|    3.491|   2|
|   CMH|     AA|     3.53|   3|
|   CMH|     DL|    4.535|   4|
|   CMH|     NW|    4.969|   5|
|   CMH|     US|     5.66|   6|
|   CMH|     YV|    7.961|   7|
|   CMH|     TW|     8.03|   8|
|   CMH|     9E|    8.326|   9|
|   CMH|     WN|     8.37|  10|
|   SRQ|     TZ|   -0.382|   1|
|   SRQ|     XE|     1.49|   2|
|   SRQ|     YV|    3.426|   3|
|   SRQ|     US|    3.489|   4|
|   SRQ|     DL|     5.31|   5|
|   SRQ|     MQ|    5.351|   6|
|   SRQ|     TW|    5.967|   7|
|   SRQ|     FL|    6.061|   8|
|   SRQ|     NW|    6.443|   9|
|   SRQ|     CO|    9.286|  10|
|   BOS|     TZ|    3.064|   1|
|   BOS|     NW|    6.951|   2|
|   BOS|     DL|    7.162|   3|
|   BOS|     EV|    7.208|   4|
|   BOS|     US|    7.941|   5|
|   BOS|     XE|    8.987|   6|
|   BOS|     AA|    9.178|   7|
|   BOS|     UA|    9.577|   8|
|   BOS|     B6|   10.046|   9|
|   BOS|     FL|   10.131|  10|
|   SEA|     OO|    2.877|   1|
|   SEA|     YV|    5.122|   2|
|   SEA|     US|    5.852|   3|
|   SEA|     DL|    6.188|   4|
```

SEA	TZ	6.345	5
SEA	NW	7.009	6
SEA	CO	7.031	7
SEA	HA	7.133	8
SEA	EV	7.896	9
SEA	AA	7.951	10
JFK	UA	5.63	1
JFK	CO	7.656	2
JFK	XE	8.114	3
JFK	DH	8.743	4
JFK	NW	11.032	5
JFK	B6	11.229	6
JFK	TW	11.496	7
JFK	DL	11.728	8
JFK	US	11.924	9
JFK	AA	11.939	10

2.2

origin	**destination**	**avg_delay**	**rank**
CMH	AUS	-5.0	1
CMH	OMA	-5.0	2
CMH	SYR	-5.0	3
CMH	CLE	0.526	4
CMH	MSN	1.0	5
CMH	SLC	3.555	6
CMH	CLT	3.667	7
CMH	IAD	4.158	8
CMH	MEM	4.295	9
CMH	IAH	4.381	10
SRQ	IAH	-0.688	1
SRQ	TPA	-0.56	2
SRQ	EYW	0.0	3
SRQ	DFW	1.858	4
SRQ	FLL	2.0	5
SRQ	MCO	2.066	6
SRQ	RSW	2.199	7
SRQ	CLE	2.462	8
SRQ	MDW	2.638	9
SRQ	CLT	2.915	10
BOS	SWF	-5.0	1
BOS	ONT	-3.0	2
BOS	GGG	1.0	3
BOS	AUS	1.324	4

```
|   BOS|        LGA|    2.749|   5|
|   BOS|        MSY|    3.763|   6|
|   BOS|        LGB|    5.431|   7|
|   BOS|        DCA|    5.538|   8|
|   BOS|        MKE|    5.945|   9|
|   BOS|        MDW|    6.003|  10|
|   SEA|        EUG|      0.0|   1|
|   SEA|        PIH|      1.0|   2|
|   SEA|        PSC|    2.651|   3|
|   SEA|        CVG|    3.571|   4|
|   SEA|        MEM|      4.0|   5|
|   SEA|        CLE|    4.749|   6|
|   SEA|        PIT|     5.13|   7|
|   SEA|        LIH|    5.381|   8|
|   SEA|        SNA|    5.649|   9|
|   SEA|        CLT|     5.68|  10|
|   JFK|        SWF|    -10.5|   1|
|   JFK|        STX|     -2.0|   2|
|   JFK|        ABQ|      0.0|   3|
|   JFK|        ANC|      0.0|   4|
|   JFK|        ISP|      0.0|   5|
|   JFK|        MYR|      0.0|   6|
|   JFK|        BGR|     3.86|   7|
|   JFK|        BQN|     3.95|   8|
|   JFK|        CHS|    4.403|   9|
|   JFK|        AGS|     4.75|  10|
+------+-----------+---------+----+
```

2.3

```
+------+-----------+---------+-------+----+
|origin|destination|avg_delay|carrier|rank|
+------+-----------+---------+-------+----+
|   BOS|        LGA|    0.663|     US|   1|
|   BOS|        LGA|     2.16|     DL|   2|
|   BOS|        LGA|   12.482|     MQ|   3|
|   BOS|        LGA|     25.6|     AA|   4|
|   BOS|        LGA|   30.448|     OH|   5|
|   BOS|        LGA|    133.0|     TZ|   6|
|   LGA|        BOS|   -3.168|     US|   1|
|   LGA|        BOS|    1.167|     DL|   2|
|   LGA|        BOS|    9.471|     MQ|   3|
|   LGA|        BOS|   27.985|     OH|   4|
|   LGA|        BOS|     40.0|     AA|   5|
|   MSP|        ATL|    5.378|     OO|   1|
|   MSP|        ATL|    6.021|     DL|   2|
```

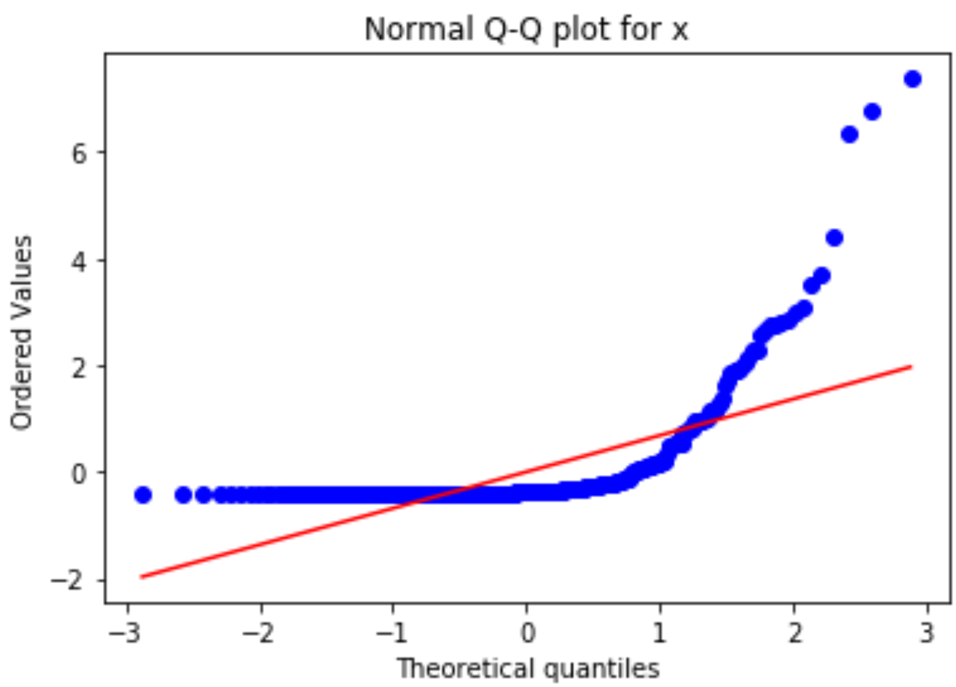
MSP	ATL	6.398	FL	3
MSP	ATL	7.74	NW	4
MSP	ATL	8.352	OH	5
MSP	ATL	10.287	EV	6
OKC	DFW	1.359	EV	1
OKC	DFW	4.049	AA	2
OKC	DFW	4.71	MQ	3
OKC	DFW	12.835	OO	4
OKC	DFW	13.401	DL	5
OKC	DFW	47.5	OH	6

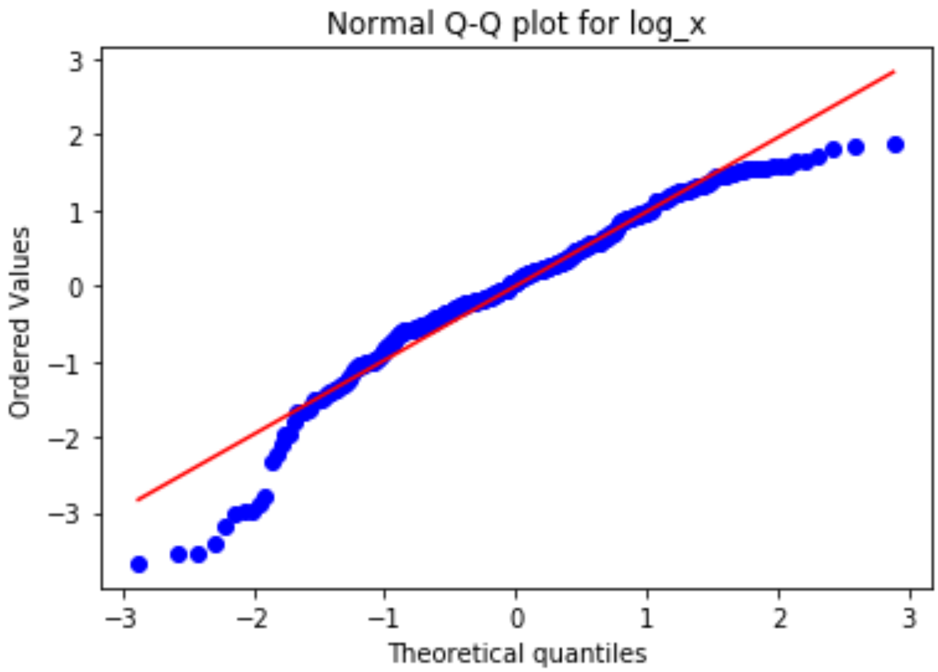
2.4

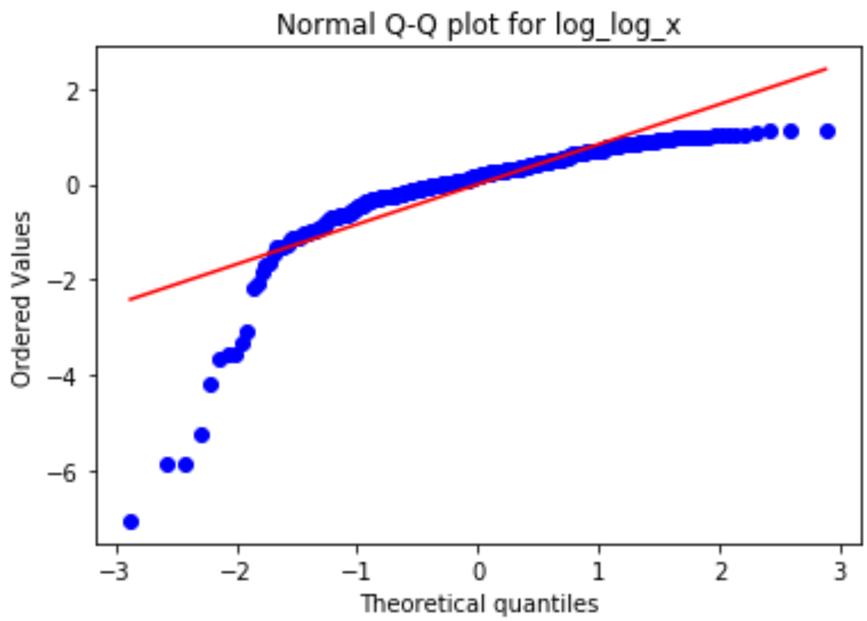
origin	**destination**	**avg_delay**
BOS	LGA	3.112
LGA	BOS	0.952
MSP	ATL	6.95
OKC	DFW	4.372

3.1

Let us compare the three Quantile-Quantile plots below (drawn in python with the **scipy.stats** package). It appears that the distribution that is closest to a normal distribution isn't that of "Log(Log(X))" but "Log(X)", which allows us to conclude that X doesn't follow Zipf 's law.




Normal Q-Q plot for log_x
Ordered Values
Theoretical quantiles
3
2
1
0
−1
−2
−3


Normal Q-Q plot for log_log_x
Ordered Values
Theoretical quantiles
2
0
−2
−4
−6
−3
−2
−1
0
1
2
3

3.2 In order, the columns are X, Y, Z, first leg's flight date, first leg's airline, first leg's flight number, and then the same pattern applies to the second leg of the trip:

```
+---+---+---+----------+----------+---------+----------+----------+--------
-+-----------+
|  x|  y|  z|xy_fl_date|carrier_xy|flight_xy|yz_fl_date|carrier_yz|flight_y
z|total_delay|
+---+---+---+----------+----------+---------+----------+----------+--------
-+-----------+
|BOS|ATL|LAX|    4/3/08|        FL|      270|    4/5/08|        FL|       4
0|          5|
|PHX|JFK|MSP|    9/7/08|        B6|      178|    9/9/08|        NW|      60
9|        -42|
|DFW|STL|ORD|   1/14/08|        AA|     1336|   1/16/08|        AA|     224
5|        -19|
|LAX|MIA|LAX|   5/16/08|        AA|      280|   5/18/08|        AA|      45
6|         -9|
+---+---+---+----------+----------+---------+----------+----------+--------
-+-----------+
```

Resources:

- Docker image: https://hub.docker.com/r/yannael/kafka-sparkstreaming-cassandra
- Zipf's law: https://en.wikipedia.org/wiki/Zipf%27s_law
- Q-Q plots: https://en.wikipedia.org/wiki/Q%E2%80%93Q_plot
- GitHub Repository: https://github.com/ErwFoui/CloudComputingProject