Cloud Computing Project

Capstone – Batch Processing

Erwan Fouillen

1. Introduction

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

- Hadoop Distributed File System
- MapReduce paradigm
- Parallelized SQL (in our case, with Hive)
- Containerized applications (use of Docker to run Hadoop, Hive & PrestoDB)

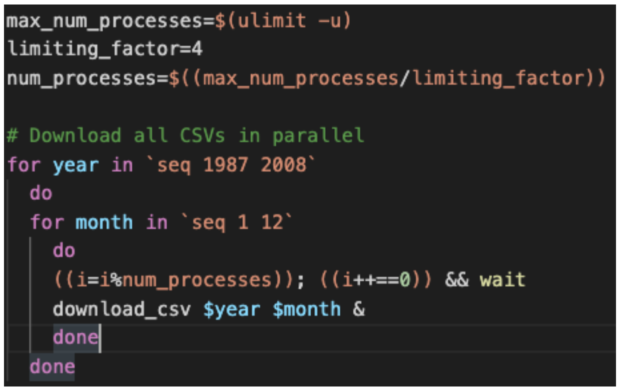
Please note that, as I had issues with AWS credits, I ended up doing everything locally with Docker, while still meeting the assignment's criteria:

1. Use HDFS to store the data in a distributed fashion (see docker image in resources)
2. Query the data with SQL statements (Hive)
3. Store the results in a database (in my case, PrestoDB)
4. 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 32 * 12 csv files are downloaded in parallel by forking bash processes (see **get_data.sh)**:



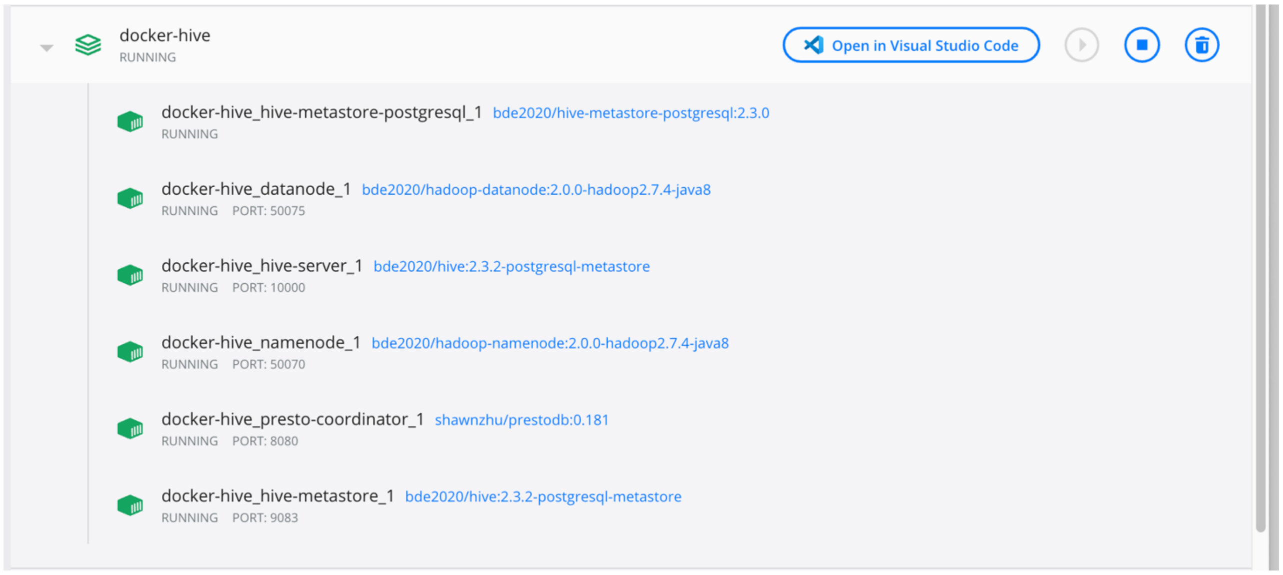
```
max_num_processes=$(ulimit -u)
limiting_factor=4
num_processes=$((max_num_processes/limiting_factor))

# 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 moved to the Hadoop namenode and added to the HDFS (see **setup_hdfs_hive.sh**)
6. In Hive, data is partitioned by **year** and **month** to increase computational efficiency. Optimizations to Hive are presented later on.

3. Architecture

All in all, the architecture is very similar to that offered by AWS, i.e., use of Hadoop to distribute files across several nodes, then use Hive to query these files with SQL statements instead of Java statements, and finally store tables in a database (PrestoDB). Everything was done locally with the help of a docker-hive image with a PrestoDB database: https://hub.docker.com/r/bde2020/hive/.
When docker is running, the following nodes are active:



docker-hive
RUNNING

docker-hive_hive-metastore-postgresql_1 bde2020/hive-metastore-postgresql:2.3.0
RUNNING

docker-hive_datanode_1 bde2020/hadoop-datanode:2.0.0-hadoop2.7.4-java8
RUNNING PORT: 50075

docker-hive_hive-server_1 bde2020/hive:2.3.2-postgresql-metastore
RUNNING PORT: 10000

docker-hive_namenode_1 bde2020/hadoop-namenode:2.0.0-hadoop2.7.4-java8
RUNNING PORT: 50070

docker-hive_presto-coordinator_1 shawnzhu/prestodb:0.181
RUNNING PORT: 8080

docker-hive_hive-metastore_1 bde2020/hive:2.3.2-postgresql-metastore
RUNNING PORT: 9083

All a user has to do is to start this docker image, run **get_data.sh** to concurrently download all the files, then run **setup_hdfs_hive.shhttps://en.wikipedia.org/wiki/Q%E2%80%93Q_ploth** to add the data to the hdfs and load them in a hive table named **“flights”**. Finally, all assignment-related queries are shown with their results in the **questions_queries.sql** file.

4. Optimizations

Based on the best practices listed at https://www.qubole.com/blog/hive-best-practices/, I implemented the following optimizations:

```
SET hive.cli.print.header=true;
SET hive.exec.parallel=true;
SET hive.compute.query.using.stats=true;
SET hive.stats.fetch.column.stats=true;
SET hive.stats.fetch.partition.stats=true;
SET hive.vectorized.execution.enabled=true;
SET hive.auto.convert.join=true;
```

In particular, 3 optimizations greatly improved performance: **hive.exec.parallel** enables parallelized execution of hive queries, **hive.vectorized.execution.enabled** enables processing of multiple rows at a time, and **hive.auto.convert.join** to automatically perform joins on small tables. Combined, these optimizations result in a ~2x speed-up.

5. Results

This last part shows answers to the assignment’s questions, with performance comparison when activating some hive optimizations. All queries and results are shown in the **questions_queries.sql** file. 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

```
orig.origin all_flights
"ORD" 13285286
"ATL" 12231578
"DFW" 11501498
"LAX" 8201586
"PHX" 7011933
"DEN" 6682815
"DTW" 5998100
"IAH" 5791682
"MSP" 5540681
"SFO" 5478875
Time taken: 277.546 seconds --> Time taken: 89.121 seconds with
hive.vectorized.execution.enabled=true and hive.auto.convert.join=true
```

1.2

```
airline avg_arr_delay
"HA"  -0.775
"KH"  1.157
"ML (1)"  4.748
"PA (1)"  5.434
"NW"  5.491
"WN"  5.515
"F9"  5.693
"OO"  5.877
"9E"  6.108
"TZ"  6.129
Time taken: 139.859 seconds --> Time taken: 80.228 seconds with
hive.vectorized.execution.enabled=true and hive.auto.convert.join=true
```

1.3

```
dow avg_arr_delay
Saturday 4.191
Tuesday 5.966
Sunday 6.529
Monday 6.684
Wednesday 7.111
Thursday 8.951
Friday 9.611
Time taken: 134.044 seconds --> Time taken: 78.216 seconds with
hive.vectorized.execution.enabled=true and hive.auto.convert.join=true
```

2.1

- **BWI**

```
"BWI" "F9"  0.756 1
"BWI" "PA (1)"  4.762 2
```

```
"BWI" "CO"  5.178 3
"BWI" "YV"  5.355 4
"BWI" "NW"  5.578 5
"BWI" "AA"  5.922 6
"BWI" "9E"  6.747 7
"BWI" "US"  7.494 8
"BWI" "UA"  7.605 9
"BWI" "FL"  7.695 10
```

- **CMI**

```
"CMI" "OH"  0.612 1
"CMI" "US"  2.033 2
"CMI" "TW"  3.848 3
"CMI" "PI"  4.599 4
"CMI" "DH"  6.028 5
"CMI" "EV"  6.665 6
"CMI" "MQ"  7.849 7
```

- **IAH**

```
"IAH" "NW"  3.485 1
"IAH" "PI"  3.825 2
"IAH" "PA (1)"  3.985 3
"IAH" "US"  4.912 4
"IAH" "AA"  5.548 5
"IAH" "F9"  5.598 6
"IAH" "TW"  5.979 7
"IAH" "EA"  6.16  8
"IAH" "WN"  6.269 9
"IAH" "MQ"  6.713 10
```

- **LAX**

```
"LAX" "MQ"  2.356 1
"LAX" "OO"  4.182 2
"LAX" "TZ"  4.764 3
"LAX" "FL"  4.872 4
"LAX" "NW"  5.101 5
"LAX" "YV"  5.89  6
"LAX" "F9"  5.987 7
"LAX" "HA"  6.116 8
"LAX" "US"  6.681 9
"LAX" "AA"  6.964 10
```

- **MIA**

```
"MIA" "EV"  1.203 1
"MIA" "TZ"  1.782 2
```

```
"MIA" "XE"  1.919 3
"MIA" "PA (1)"  4.19  4
"MIA" "NW"  4.443 5
"MIA" "US"  6.044 6
"MIA" "UA"  6.662 7
"MIA" "ML (1)"  7.505 8
"MIA" "PI"  8.064 9
"MIA" "MQ"  8.547 10
```

- **SFO**

```
"SFO" "TZ"  3.952 1
"SFO" "MQ"  4.975 2
"SFO" "PA (1)"  5.278 3
"SFO" "F9"  5.447 4
"SFO" "NW"  5.592 5
"SFO" "DL"  6.474 6
"SFO" "CO"  7.106 7
"SFO" "US"  7.434 8
"SFO" "TW"  7.743 9
"SFO" "AA"  7.836 10
Time taken: 0.073 seconds, Fetched: 57 row(s)
```

2.2

- **BWI**

```
"BWI" "SAV" -7.0  1
"BWI" "MLB" 1.155 2
"BWI" "DAB" 1.47  3
"BWI" "SRQ" 1.558 4
"BWI" "IAD" 1.597 5
"BWI" "UCA" 3.654 6
"BWI" "CHO" 4.012 7
"BWI" "GSP" 4.198 8
"BWI" "MDT" 4.421 9
"BWI" "OAJ" 4.494 10
```

- **CMI**

```
"CMI" "ABI" -7.0  1
"CMI" "PIT" 1.102 2
"CMI" "CVG" 1.895 3
"CMI" "DAY" 3.256 4
"CMI" "STL" 3.727 5
"CMI" "PIA" 4.163 6
"CMI" "DFW" 6.334 7
```

```
"CMI" "ATL" 6.665 8
"CMI" "ORD" 7.939 9
"CMI" "SPI" 587.0 10
```

- **IAH**

```
"IAH" "MSN" -2.0  1
"IAH" "AGS" -0.619  2
"IAH" "MLI" -0.5  3
"IAH" "EFD" 1.888 4
"IAH" "HOU" 2.091 5
"IAH" "JAC" 2.388 6
"IAH" "SBA" 3.0 7
"IAH" "RNO" 3.161 8
"IAH" "MTJ" 3.175 9
"IAH" "BPT" 3.6 10
```

- **LAX**

```
"LAX" "SDF" -16.0 1
"LAX" "IDA" -7.0  2
"LAX" "DRO" -6.0  3
"LAX" "RSW" -3.0  4
"LAX" "LAX" -2.0  5
"LAX" "BZN" -0.231  6
"LAX" "MAF" 0.0 7
"LAX" "PIH" 0.0 8
"LAX" "IYK" 1.169 9
"LAX" "MFE" 1.376 10
```

- **MIA**

```
"MIA" "SHV" 0.0 1
"MIA" "BUF" 1.0 2
"MIA" "SAN" 1.71  3
"MIA" "SLC" 2.537 4
"MIA" "HOU" 3.145 5
"MIA" "ISP" 3.647 6
"MIA" "MEM" 3.659 7
"MIA" "PSE" 3.711 8
"MIA" "TLH" 4.235 9
"MIA" "GNV" 4.827 10
```

- **SFO**

```
"SFO" "SDF" -7.5  1
"SFO" "MSO" -4.0  2
"SFO" "PIH" -3.0  3
```

```
"SFO" "LGA" -1.758  4
"SFO" "PIE" -1.341  5
"SFO" "OAK" -0.409  6
"SFO" "FAR" 0.0 7
"SFO" "BNA" 2.426 8
"SFO" "MEM" 3.184 9
"SFO" "SJC" 4.307 10
Time taken: 0.076 seconds, Fetched: 60 row(s)
```

2.3

- **CMI – ORD**

```
"CMI" "ORD" "MQ"  9.707 1
```

- **IND – CMH**

```
"IND" "CMH" "CO"  -2.549  1
"IND" "CMH" "HP"  5.316 2
"IND" "CMH" "AA"  5.5 3
"IND" "CMH" "NW"  5.762 4
"IND" "CMH" "US"  7.136 5
"IND" "CMH" "DL"  10.688  6
"IND" "CMH" "EA"  12.407  7
```

- **DFW – IAH**

```
"DFW" "IAH" "PA (1)"  -1.596  1
"DFW" "IAH" "UA"  3.502 2
"DFW" "IAH" "EV"  5.093 3
"DFW" "IAH" "CO"  6.52  4
"DFW" "IAH" "OO"  7.564 5
"DFW" "IAH" "XE"  8.066 6
"DFW" "IAH" "AA"  8.12  7
"DFW" "IAH" "DL"  8.669 8
"DFW" "IAH" "MQ"  9.103 9
```

- **LAX – SFO**

```
"LAX" "SFO" "TZ"  -7.619  1
"LAX" "SFO" "F9"  -2.029  2
"LAX" "SFO" "EV"  6.965 3
"LAX" "SFO" "AA"  7.553 4
"LAX" "SFO" "US"  7.727 5
"LAX" "SFO" "MQ"  7.808 6
```

```
"LAX" "SFO" "CO"  9.343 7
"LAX" "SFO" "NW"  9.849 8
"LAX" "SFO" "UA"  9.958 9
"LAX" "SFO" "WN"  10.367  10
```

- **JFK – LAX**

```
"JFK" "LAX" "B6"  NULL  1
"JFK" "LAX" "UA"  2.984 2
"JFK" "LAX" "AA"  6.648 3
"JFK" "LAX" "HP"  6.681 4
"JFK" "LAX" "DL"  7.385 5
"JFK" "LAX" "PA (1)"  11.294  6
"JFK" "LAX" "TW"  11.697  7
```

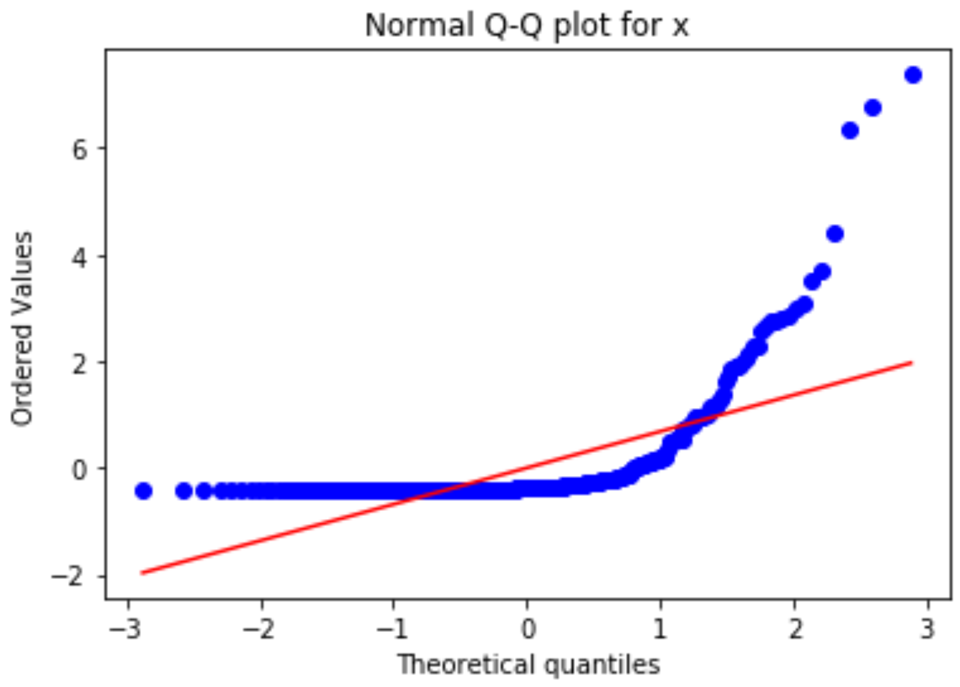
- **ATL – PHX**

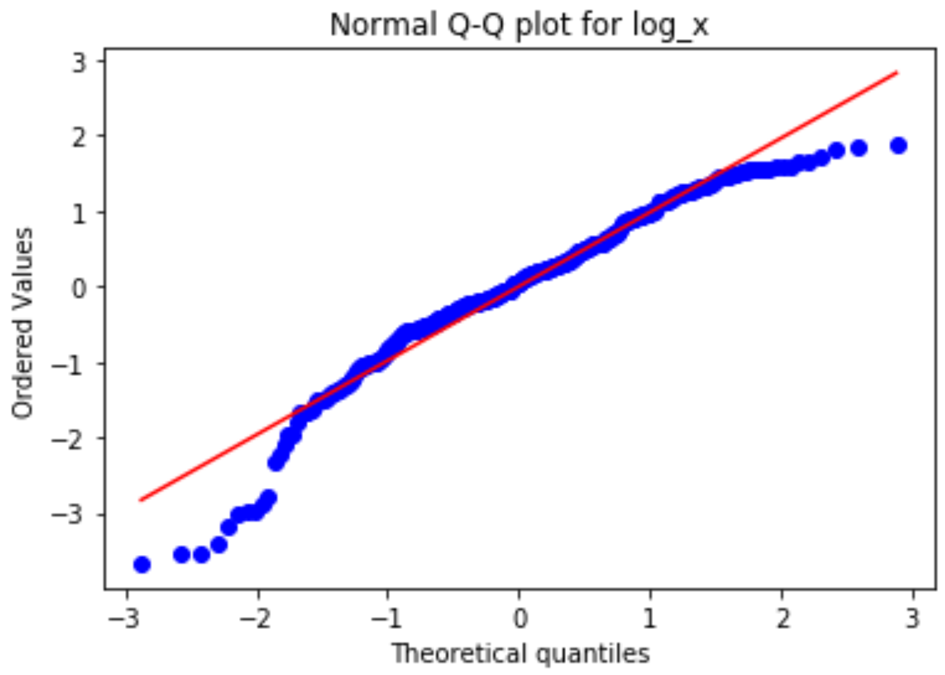
```
"ATL" "PHX" "FL"  5.023 1
"ATL" "PHX" "US"  6.207 2
"ATL" "PHX" "HP"  8.071 3
"ATL" "PHX" "DL"  9.926 4
"ATL" "PHX" "EA"  10.335  5
```

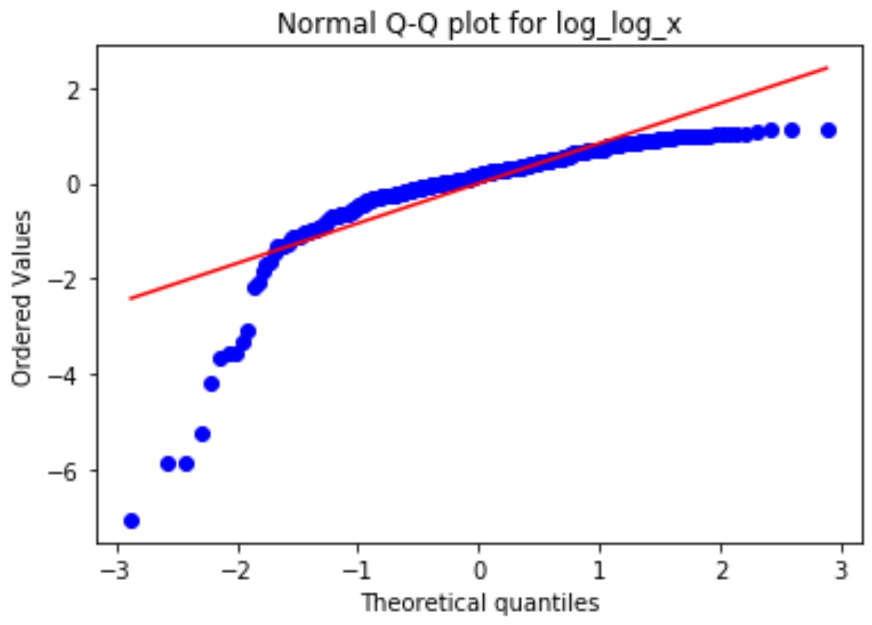
2.4

```
"ATL" "PHX" 9.047
"CMI" "ORD" 9.707
"DFW" "IAH" 7.563
"IND" "CMH" 3.142
"JFK" "LAX" 6.339
"LAX" "SFO" 9.621
Time taken: 1.247 seconds, Fetched: 6 row(s)
```

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 x
Ordered Values
Theoretical quantiles


Normal Q-Q plot for log_x
Ordered Values
Theoretical quantiles



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

```
"CMI" "ORD" "LAX" 2008-03-04  "0710"  "MQ"  "4278"  2008-03-06  "1950"  "AA"  "607" -
38.0
"DFW" "ORD" "DFW" 2008-06-10  "0700"  "UA"  "1104"  2008-06-12  "1645"  "AA"  "2341"  -
31.0
"JAX" "DFW" "CRP" 2008-09-09  "0725"  "AA"  "845" 2008-09-11  "1645"  "MQ"  "3627"  -6.0
"LAX" "SFO" "PHX" 2008-07-12  "0650"  "WN"  "3534"  2008-07-14  "1925"  "US"  "412" -
32.0
"LAX" "ORD" "JFK" 2008-01-01  "0705"  "UA"  "944" 2008-01-03  "1900"  "B6"  "918" -6.0
"SLC" "BFL" "LAX" 2008-04-01  "1100"  "OO"  "3755"  2008-04-03  "1455"  "OO"  "5429"
18.0
Time taken: 175.682 seconds --> with hive.vectorized.execution.enabled=true and
hive.auto.convert.join=true
```

Resources:

- Hive Optimizations: https://www.qubole.com/blog/hive-best-practices/
- Docker-Hive with PrestoDB connector: https://hub.docker.com/r/bde2020/hive/
- 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