# Big Data Processing Systems
# 2nd Project

Helder Rodrigues
*MAEBD*
*FCT UNL PT*
*Email: harr@campus.fct.unl.pt*

***Abstract*—This document presents a set of experiments with the aim of showing the capabilities of 2 Big Data technologies, Hive and Spark in 2 flavours DataFrames and SQL, for extracting insights by processing a great volume of data.**

## 1. Introduction

We will go trough the Project 2 statement [1] analysing the information about taxi rides in some city for better understanding the behavior of the community and help taxi drivers maximize their profit. We will use Big Data technology during our experiments, Hive and Spark in 2 flavours Data Frames and SQL, and take conclusions about the analysis methods. For Spark we used the python API denominated PySpark.

hr
December 2, 2020

## 2. Dataset

The Dataset consists in a collection of taxi trip records in some city, including fields for capturing pick-up and drop-off dates/times, pick-up and drop-off locations and trip distances.

## 3. PySpark Data Frames

We used PySpark Data Frames to create indexes for answering the following queries:

### 3.1. How many trips were started in each year present in the data set?

With a simple script, built in fluent mode, we applied an aggregation function *count* on top of an aggregation by the year of the trip.

Table 1 shows an example of the obtained data.

The obtained performance counters were: user 5.33 ms, sys: 3.94 ms, total: 9.27 ms

TABLE 1. RESULTS OF 1.1

| year | #trips |
|---|---|
| 2013 | 54409 |
| 2014 | 74753 |
| ... | ... |
| 2019 | 32797 |
| 2020 | 6829 |

TABLE 2. RESULTS OF 1.2

| time_slot | #trips | avg_trip_miles | avg_trip_total |
|---|---|---|---|
| 05 PM | 23637 | 3.04 | 15.16 |
| 08 PM | 22218 | 3.18 | 15.41 |
| 11 AM | 18616 | 3.42 | 15.41 |
| 10 AM | 17775 | 3.20 | 15.22 |
| 04 AM | 4604 | 3.80 | 15.19 |
| ... | ... | ... | ... |
| 11 PM | 16299 | 2.86 | 14.32 |
| 06 AM | 5628 | 5.25 | 19.92 |
| 09 PM | 19783 | 3.31 | 15.44 |
| 10 PM | 18491 | 3.11 | 14.95 |
| 09 AM | 18246 | 3.10 | 14.47 |
| 03 PM | 20704 | 3.50 | 16.26 |

### 3.2. For each of the 24 hours of the day, how many taxi trips there were, what was their average trip miles and trip total cost?

This problem is solved by a simple grouping by the trip hour. In this case, instead of computing a single kpi we obtained the value of 3 aggregation functions over the same aggregation.

Table 2 shows an example of the obtained data.

The obtained performance counters were: user 16.5 ms, sys: 8.98 ms, total: 25.5 ms.

TABLE 3. RESULTS OF 1.3

| pickup_location | #trips | a_miles | a_total |
|---|---|---|---|
| 05 PM 17031839100 17031839100 | 466 | 0.58 | 6.66 |
| 05 PM 17031839100 17031281900 | 327 | 0.58 | 6.56 |
| 05 PM 17031839100 17031320100 | 292 | 0.69 | 7.20 |
| ... | ... | ... | ... |
| 05 PM 17031839100 17031280100 | 199 | 0.67 | 6.77 |
| 08 PM 17031839100 17031839100 | 183 | 0.49 | 6.17 |

TABLE 4. HISTORIC AVERAGE PRICES OF GAS IN THE U. S. [4] - MATRIX FORMAT

| Year | Jan | Feb | Mar | Apr | May | Jun | ... |
|---|---|---|---|---|---|---|---|
| 2006 | 2.36 | 2.326 | 2.468 | 2.787 | 2.953 | 2.93 | ... |
| 2007 | 2.289 | 2.323 | 2.609 | 2.891 | 3.187 | 3.102 | ... |
| 2008 | 3.095 | 3.078 | 3.293 | 3.507 | 3.815 | 4.105 | ... |
| 2009 | 1.84 | 1.975 | 2.011 | 2.102 | 2.316 | 2.681 | ... |
| 2010 | 2.769 | 2.699 | 2.824 | 2.9 | 2.89 | 2.785 | ... |
| 2011 | 3.148 | 3.264 | 3.615 | 3.852 | 3.96 | 3.735 | ... |
| 2012 | 3.44 | 3.64 | 3.907 | 3.958 | 3.791 | 3.596 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... |

### 3.3. For each of the 24 hours of the day, which are the 5 most popular routes according to the the total number of taxi trips? Also report and the average fare.

This problem requires the computation of a ranking over one key. We used Spark's Data Frames windowing functions provided by the Window class. Table 3 shows an example of the obtained data.

The obtained performance counters were: user 35.7 ms, sys: 14.5 ms, total: 50.2 ms

### 3.4. EXTRA: What's the impact of the fuel price in the ride total?

We stared by obtaining the historic prices of gas in the the U. S. [4].

Unfortunately the data wasn't in tabular format but in a matrix of years x months, like presented on Table 4. Spark Data Frames has a very handy expression *stack* for unpivoting matrices, transforming it into tables. After unpivoting the data we organised the data like shown in Table 5.

We joined this data with the taxi rides data and we could calculate the cost of the consumed gas during each trip, taking into account an average consumption of 26.3 mpg [5]. Table 6 shows an example of the obtained data.

We can conclude that although the gas price oscillated substantially it always represented slightly the same proportion in the total bill, around 2.2%.

The obtained performance counters were: user 29.4 ms, sys: 20.1 ms, total: 49.5 ms

TABLE 5. HISTORIC AVERAGE PRICES OF GAS IN THE U. S. [4] - TABULAR FORMAT

| Year | Month | Price |
|---|---|---|
| 1994 | Jan | 0.998 |
| 1994 | Feb | 1.009 |
| 1994 | Mar | 1.008 |
| 1995 | Jan | 1.13 |
| 1995 | Feb | 1.12 |
| 1995 | Mar | 1.119 |
| ... | ... | ... |

TABLE 6. RESULTS OF 1.4

| year | % gas cost on total | avg gas price $/g | $/mile |
|---|---|---|---|
| 2013 | 2.2 | 3.57 | 6.17 |
| 2014 | 2.68 | 3.46 | 4.90 |
| 2015 | 2.06 | 2.53 | 4.68 |
| 2016 | 2.02 | 2.25 | 4.26 |
| 2017 | 2.19 | 2.53 | 4.38 |
| 2018 | 2.37 | 2.83 | 4.54 |
| 2019 | 2.18 | 2.69 | 4.72 |
| 2020 | 1.98 | 2.47 | 4.70 |

TABLE 7. PERFORMANCE OF 2.1

| Hive | Spark SQL |
|---|---|
| 14.62 s | 44.1 ms |

## 4. Spark SQL and Hive

We used Spark SQL and Hive to create indexes for answering the queries presented in the subsections bellow. The implementation is almost the same for both technologies as all the business logic is implemented in the SQL query.

For hive we stared by creating a table, accordingly with the supplied csv file structure. For the columns containing date/time values we needed to instruct the serde about the regular expression of the date and time representation. For Spark SQL we also needed to indicate the regular expression of the Timestap columns, here in the read method for dataframe creation.

### 4.1. What is the accumulated number of taxi trips per month?

We developed a small SQL query for grouping the data per month and count the number or trips. The query run, without any modification, on both Hive and Spark SQL. Table 8 shows an example of the obtained data.

The obtained performance counters can be checked on Table 7

TABLE 8. RESULTS OF 2.1

| month_number | #total_trips |
|---|---|
| 12 | 29252 |
| 01 | 30357 |
| ... | ... |
| 03 | 35260 |
| 05 | 34979 |

TABLE 9. RESULTS OF 2.2

| pickup_region_ID | list_of_dropoff_region_ID |
|---|---|
| 17031770202 | ['17031770202'] |
| 17031020301 | ['17031020802', '17031020301'] |
| 17031837100 | ['17031243000', '17031320400', '17031837100'] |
| 17031030104 | ['17031030104', '17031980000', '17031280100'] |
| 17031040300 | ['17031320400', '17031040300'] |
| ... | ... |

TABLE 10. PERFORMANCE OF 2.2

| Hive | Spark SQL |
|---|---|
| 15.50 s | 190 ms |

### 4.2. For each pickup region, report the list of unique dropoff regions?

Here we used the *collect_list* aggregation function to concatenate all the aggregated lines in a single list. We also use a sub query for eliminate duplicates using the *distinct* key word.

The obtained performance counters can be checked on Table 10

### 4.3. What is the expected duration and distance of a taxi ride, given the pickup region ID, the weekday (0=Monday, 6=Sunday) and time in format "hour AM/PM"?

For presenting the weekday we haven't found a common approach for Spark SQL and for Hive. For Spark SQL we developed an UDF as the Spark *date_format* hasn't a numeric representation for the weekday. Hive has the capability to extract the numeric representation of the weekday, however it isn't aligned with the specification, so we needed to convert the weekday to integer and subtract 1.

Table 11 shows an example of the obtained data.

The obtained performance counters can be checked on Table 12. The UDF highly penalised the performance. We could try other approaches like a case condition for each textual weekday or a join with a temporary table with pairs (Mon, 0),(Tue, 1)...

TABLE 11. RESULTS OF 2.3

| pickup | avg_total_trip_cost |
|---|---|
| 17031081600_3_12_PM | 5.85 |
| 17031071500_5_12_PM | 3.25 |
| ... | ... |
| 17031081500_5_12_AM | 12.45 |
| 17031842300_1_08_AM | 43.8 |
| 17031081000_5_03_PM | 7.05 |

TABLE 12. PERFORMANCE OF 2.3

| Mode | Hive | Spark SQL |
|---|---|---|
| With Weekday (UDF for Spark) | 16.77 s | 3.85 s |
| No Weekday | 15.33 s | 4.87 ms |

### 4.4. EXTRA: Which are the top 3 companies driving from the most popular locations? (more than 5000 trips from there)

We observe that the *Taxi Affiliation Services* company dominates the trips from the most popular locations in the city followed by *Flash Cab*, the third place is shared between several smaller providers.

As this insight is complex we needed to create 4 subqueries using the SQL **with** statement. For being able to obtaining the 3 ranking measures, top 3 and bugger than 5000 trips, we needed to use **windowing functions**.

Even with complex queries like this one the syntax on Hive and on Spark SQL was exactly the same.

The obtained performance counters can be checked on Table 14

## 5. Conclusion

- HIVE language is designated HQL, its syntax isn't filly SQL compliant however in our case we haven't found any particular limitation.
- In Big Data, like in most of the computing frameworks, working with dates and times is often tricky. Most of the particularities were solved at the beginning during the Table creation for Hive and during the first dataframe instantiation for both Spark technologies.

### 5.1. Performance

- We collected the ELAPSED TIME measure of Hive and the total CPU time for Spark. This means we don't consider the time for rendering results, as we have different outputs CLI and Jupyter Notebook.
- For being able to measure the computation of the full DAG of our Spark exercises we executed a *cache()* action. Otherwise the *show()* function would only compute the necessary to show an output sample.

TABLE 13. RESULTS OF 1.4

| pickupregionid | company | ntrips |
|---|---|---|
| 17031839100 | Taxi Affiliation Services | 8485 |
| 17031839100 | Flash Cab | 3485 |
| 17031839100 | Yellow Cab | 2308 |
| 17031320100 | Taxi Affiliation Services | 4567 |
| 17031320100 | Flash Cab | 1768 |
| 17031320100 | Blue Ribbon Taxi Association Inc. | 1396 |
| 17031980000 | Taxi Affiliation Services | 2955 |
| 17031980000 | Flash Cab | 1404 |
| 17031980000 | Dispatch Taxi Affiliation | 950 |
| 17031081500 | Taxi Affiliation Services | 3452 |
| 17031081500 | Flash Cab | 1201 |
| 17031081500 | Choice Taxi Association | 1002 |
| 17031081700 | Taxi Affiliation Services | 3110 |
| 17031081700 | Flash Cab | 1476 |
| 17031081700 | Dispatch Taxi Affiliation | 997 |
| 17031281900 | Taxi Affiliation Services | 2736 |
| 17031281900 | Chicago Carriage Cab Corp | 1063 |
| 17031281900 | Flash Cab | 891 |
| 17031081403 | Taxi Affiliation Services | 2313 |
| 17031081403 | Flash Cab | 814 |

TABLE 14. PERFORMANCE OF 2.4

| Hive | Spark SQL |
|---|---|
| 12.67 s | 6.13 ms |

This was valid for Spark Data frames as well as for Spark SQL.

- UDF functions in Spark SQL highly penalise performance.
- Hive performance is much poor, in most cases, when compared with Spark SQL. However this isn't always true as today Hive has engines, other than MapReduce or Tez, like LLAP that promise very good results for interactive querying.

## References


[1] J. Lourenço, *Big Data Processing Systems — Project nº 2*, 2020/21. DI, FCT, UNL, PT.

[2] J. Lourenço, *Big Data Processing Systems — Class Lectures*, 2020/21. DI, FCT, UNL, PT.

[3] Thomas Erl, Wajid Khattak and Paul Buhler *Big Data Fundamentals*, 2016 Prentice Hall.

[4] U.S. Energy Information Administration *U.S. All Grades All Formulations Retail Gasoline Prices (Dollars per Gallon)*

[5] Wikipedia *Fuel economy in automobiles*


## Work division

59290 - Helder Rodrigues - All