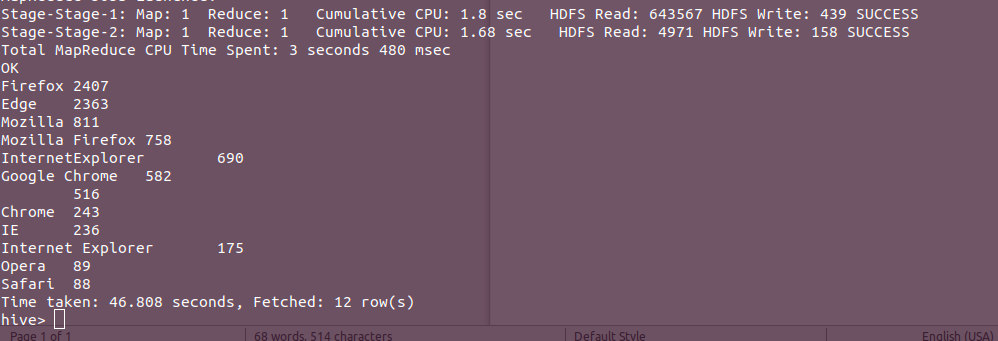
Hive Pig Spark Comparison

In e-commerce, it is vital for the businesses to know the preferences and behaviors of the users so that businesses can make informed decisions. This is achieved by keeping track of all the users’ cursor clicks on the sites. The data obtained is the clickstream data. This section would focus on the clickstream data using different Hadoop tools.

The sample dataset is in the structure of id, date time, site id, offer id, category, merchant, country code, browser id, and dev id. From the raw data, it is found that there are some redundancies in the browser id column as shown in the figure below. For example, the same Mozilla Firefox browser has 3 different names, namely Mozilla, Firefox and Mozilla Firefox. This redundancies should be cleansed to obtain an accurate count for each of the browser.



The following section would show the different tools cleansing the browserid column and aggregate the count of each browser.

# Apache Hive

## Loading data

In Hive, a table must be created before importing the data into the table.

The command to create the table in Hive:

*create table clickstream(id string, datetime string, siteid string, offerid string, category string, merchant string, countrycode string, browserid string, devid string) row format delimited fields terminated by ',';*

Importing the data from local:

*load data local inpath '/home/dante/Downloads/clickdata.csv' overwrite into table clickstream;*

Importing the data from HDFS:

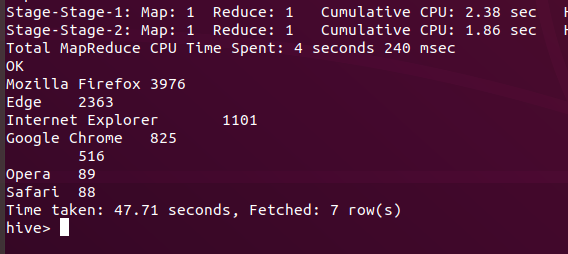
*load data inpath ‘/user/dante/clickdata.csv’ into table clickstream*

## Cleanse and Aggregate

Apache Hive is a data warehouse. Data warehouse is for one time loading and multiple reading, no editing is allowed. Therefore, it does not support updating or editing of the values. However, we can obtain the cleansed data without editing the table. The following Hive commands cleanse the data and aggregate the data.



The result of the command is shown below:



Both cleansing and aggregating the data took Hive 47.71 seconds.

# Apache Pig

## Loading data

In Pig, the data can be read directly from HDFS instead of creating a table as Hive, but the structure of the data also need to be specified when the data is loaded. The Pig command to load the data:

*streamed\_data = LOAD '/user/dante/clickdata.csv' USING PigStorage(',') AS (id:chararray, datetime:chararray, siteid:chararray, offerid:chararray, category:chararray, merchant:chararray, countrycode:chararray, browserid:chararray, devid:chararray);*

## Cleanse and Aggregate

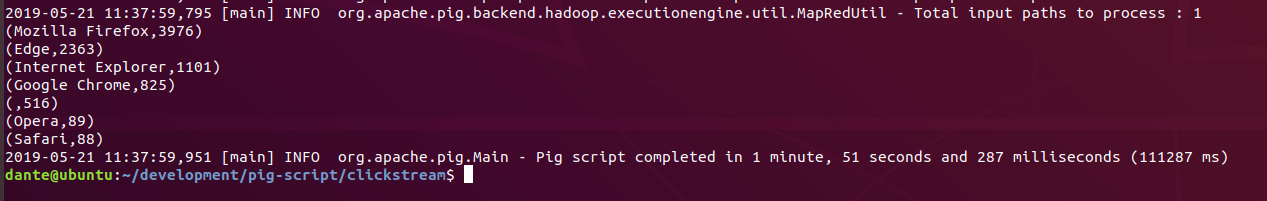
As shown above, the redundancies of the browser id are cleansed in order to obtain an accurate count. The commands of Pig to load, cleanse and aggregate the data is shown as follow. The cleansing process in Pig is more complex than Hive. Each row of the data cleansed would remain separated from the full data. After all the redundancies in the data are cleansed, the rows have to be joined back to the original dataset to form the cleansed data.

In the joining process, the original browserid column is dropped since it has redundancies, the new browserid column would be the cleansed data. The cleansed data can be stored into HDFS and aggregation can be performed by reading the cleansed data from HDFS or it can continue to use the cleansed data for aggregation without storing.

With the cleansed data, the aggregation can be performed on the column to get an accurate count of the browsers.



The output of the aggregation:



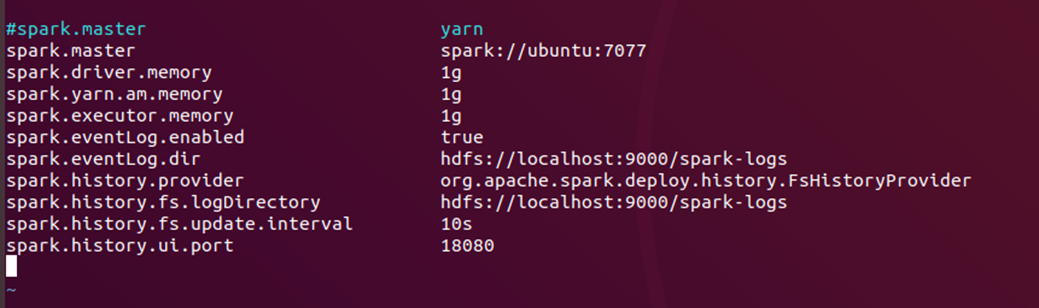
Pig took 1 minute and 51 seconds just to cleanse and aggregate the data.

# Apache Spark

Apache Spark is a fast in-memory data processing engine that allows data workers to execute streaming, machine learning or SQL workloads efficiently. Spark can integrate with many different languages, including Java, Scala, Python, R and SQL. (Hortonworks, 2018)

## Installation

1. Download the tar.gz file from <https://spark.apache.org/downloads.html>
2. Extract the file and move it to a desired location (e.g. /home/user/)
3. Install Scala (sudo apt install scala)
4. Include spark/bin and scala/bin directory into the PATH variable ( .bashrc)
5. Edit the conf/spark-defaults.conf to configure Spark



1. Start spark-master, spark-slaves and history server
2. Start spark shell with Scala with the command: spark-shell –master spark:<ip>:<port>
3. Or start spark shell with Python with the command: pyspark –master spark:<ip>:<port>

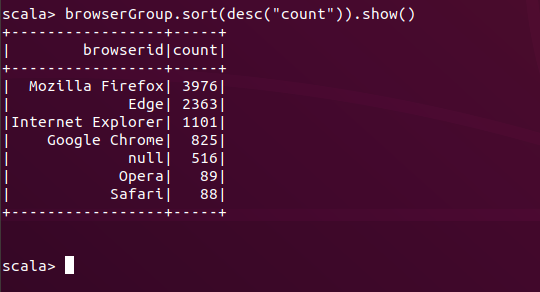
Apache Spark shell has 2 variants namely Scala shell and Python Shell. The syntax for each of them varied slightly. It is mentioned that Spark with Scala is more efficient than Python. This is because Scala is statically typed as Python is dynamically typed, which reduces the speed. Scala can run on JVM. The Spark libraries for Python requires a lot of core processing and Python does not support true multithreading so it would be slower. However, the Python syntax is easier to be understand compare to Scala, the learning curve on Scala is higher than Python. Apache Spark is written in Scala, so the latest features would usually be available to Scala first. The language choice should depend on the features needed by the project. (Gandhi, 2018)

## Spark Shell (Scala)

### Cleanse and Aggregation



The cleansed data is stored into HDFS.





The screenshot above show that Spark-shell took 27s to cleanse and aggregate the data.

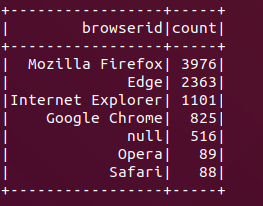
## Pyspark (Python)

### Cleanse and Aggregate

The shell command for Pyspark to cleanse and aggregate the data is shown below:



The output of the command:



The information from the spark master status page:



It is shown that cleansing and aggregating the data with Pyspark took 25s.

## Python Program

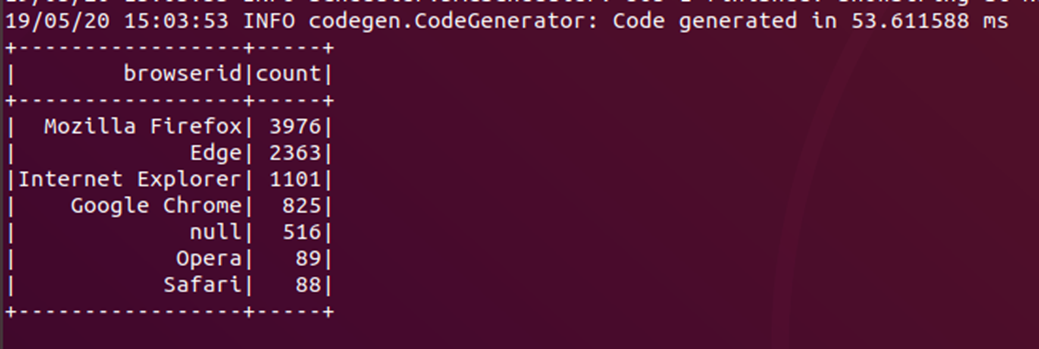
Besides, interacting with Spark via the shell, it can run a program as well. A Python program is written to perform the same task.

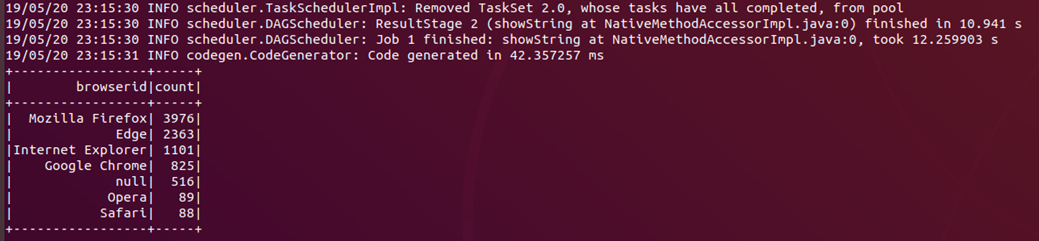


The Python program is executed via Spark with the command:

*spark-submit cleansedBrowser.py*

The output of the program above:





The information from the spark master status page:



The program ran twice in a row. The time taken for 1 complete run is 28s.

# Discussion

To summarise:

|  |  |
| --- | --- |
| Tools | Time taken (s) |
| Hive | 47.71 |
| Pig | 111 |
| Spark shell (Scala) | 27 |
| Pyspark | 25 |
| Spark python program | 28 |

Both Hive and Pig ran on top of MapReduce frameworks and utilizes MapReduce framework to process the data. From the Pig vs Hive benchmark test of IBM, it is found that Pig outperform Hive in almost every scenario except in grouping data. This result contradicts with the existing literature, Apache’s own performance benchmark, which found that Pig is significantly slower than Hive. (Jakobus & McBrien, 2013)

In the scenario of cleansing and aggregating the data above, Pig used the “group by” operator which is a scenario that Pig is slow at. In the Apache benchmark, Pig outperformed Hive in all instances except when using the “group by” operator. Pig is 104% slower than Hive in grouping data. This might be the reason why Pig took more time than Hive to generate the aggregation.

Pig is using Pig Latin scripting language which has a steep learning curve while Hive is using HQL which is similar to SQL, therefore Hive is easier to use than Pig.

Spark is not dependent on MapReduce framework. Therefore, it is not limited by the limitations of MapReduce. In MapReduce, the output from reducers must be serialized back to HDFS, it cannot be retained in memory for further processing. Complex algorithms have to be expressed in the form of Mappers and Reducers, while it is doable, experience programmers desire to reason in higher level data analysis abstractions, rather than in Mappers and Reducers. (LinuxFoundationX, 2019)

Spark overcame both challenges and even considered to be a replacement of MapReduce. Spark is built for in-memory distributed computing use cases, it implemented the concepts from MapReduce in a new way.

Since Pig and Hive are still using MapReduce, Spark outperform them which is shown in the table above. Spark is built in Scala thus Scala should be able to get a slight edge over Python. (Gandhi, 2018).

The cleansing and aggregation performed in this use case is trivial compare to the true scale and process of a large Hadoop and Spark clusters. The results are does not do the Hadoop clusters and Spark cluster justice, but it does compare the basic differences and the performance of the tools.

# Bibliography

Gandhi, P. (May, 2018). *Apache Spark: Python vs Scala*. Retrieved from Kdnuggets: https://www.kdnuggets.com/2018/05/apache-spark-python-scala.html

Hortonworks. (2018). *Apache Spark*. Retrieved from Hortonworks: https://hortonworks.com/apache/spark/

Jakobus, B., & McBrien, P. (2013). *Pig vs Hive: Benchmarking High Level Query Languages.* IBM.

LinuxFoundationX. (2019). *Shortcomings of MapReduce*. Retrieved from edx: https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS103x+2T2017/courseware/998ee852ea4142b2b51eb9bd7a027263/58c65932d30a4428a81c6668e626fbbc/?child=first