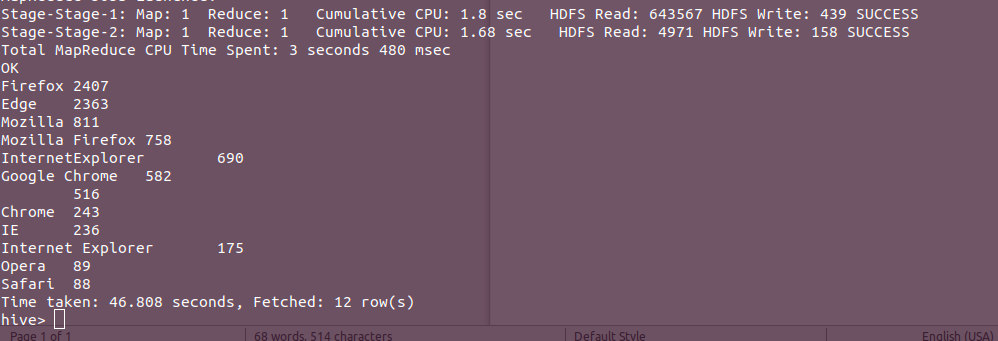
Hive Pig Spark Comparison

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. The same Mozilla Firefox browser is split into 3 different names, namely Mozilla, Firefox and Mozilla Firefox. This redundancies should be cleansed to obtain an accurate count.



The following section would show the different tools cleansing this column and aggregate the data based on browserid.

# 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.

*SELECT cleansed.n\_browserid, COUNT(\*) AS freq FROM (SELECT \*, CASE WHEN click.browserid = 'Mozilla' THEN "Mozilla Firefox"*

*WHEN click.browserid = 'Firefox' THEN 'Mozilla Firefox'*

*WHEN click.browserid = 'Mozilla Firefox' THEN 'Mozilla Firefox'*

*WHEN click.browserid = 'InternetExplorer' THEN 'Internet Explorer'*

*WHEN click.browserid = 'IE' THEN 'Internet Explorer'*

*WHEN click.browserid = 'Internet Explorer' THEN 'Internet Explorer'*

*WHEN click.browserid = 'Chrome' THEN 'Google Chrome'*

*WHEN click.browserid = 'Google Chrome' THEN 'Google Chrome'*

*WHEN click.browserid = 'Edge' THEN 'Edge'*

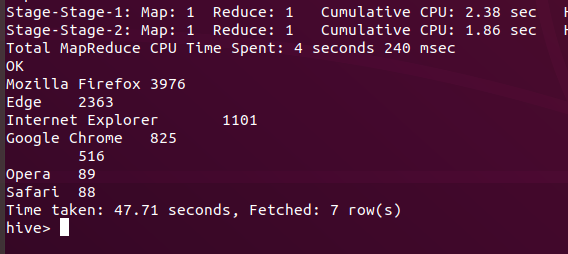
*WHEN click.browserid = 'Opera' THEN 'Opera'*

*WHEN click.browserid = 'Safari' THEN 'Safari'*

*ELSE ''*

*END AS n\_browserid FROM clickstream click) cleansed GROUP BY cleansed.n\_browserid ORDER BY freq DESC;*

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 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 the data

As shown above, the redundancies of the browser id is cleansed in order to obtain an accurate count.

*-- Convert the datetime from string to datetime format*

*datetime\_col = FOREACH streamed\_data GENERATE id, ToDate($1, 'dd/MM/yyyy HH:mm');*

*-- Cleansing the redundancy in browserid column*

*cleansed\_firefox = FOREACH streamed\_data GENERATE id, REPLACE(browserid, '^Firefox$', 'Mozilla Firefox') AS browserid;*

*cleansed\_mozilla = FOREACH cleansed\_firefox GENERATE id, REPLACE(browserid, '^Mozilla$', 'Mozilla Firefox') AS browserid;*

*cleansed\_inte = FOREACH cleansed\_mozilla GENERATE id, REPLACE(browserid, '^InternetExplorer$', 'Internet Explorer') AS browserid;*

*cleansed\_ie = FOREACH cleansed\_inte GENERATE id, REPLACE(browserid, '^IE$', 'Internet Explorer') AS browserid;*

*cleansed\_chrome = FOREACH cleansed\_ie GENERATE id, REPLACE(browserid, '^Chrome$', 'Google Chrome') AS browserid;*

*-- Join the columns back together*

*join\_data = JOIN streamed\_data by id, datetime\_col by id, cleansed\_chrome by id;*

*cleansed\_data = FOREACH join\_data GENERATE $0 AS id, $10 AS datetime, $2 AS siteid, $3 AS offerid, $4 AS category, $5 AS merchant, $6 AS countrycode, $12 AS browserid, $8 AS devid;*

*STORE cleansed\_data INTO '/user/dante/cleansed\_clickstream' USING PigStorage(',');*

The cleansed data from Pig is stored into HDFS.

## Aggregation

The command in Pig to aggregate the data

*-- group the data by browser*

*streamed\_data = LOAD '/user/dante/cleansed\_clickstream/part-r-00000' USING PigStorage(',') AS (id:chararray, datetime:chararray, siteid:chararray, offerid:chararray, category:chararray, merchant:chararray, countrycode:chararray, browserid:chararray, devid:chararray);*

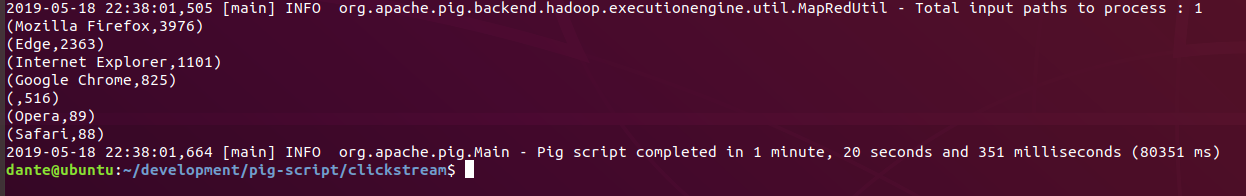
*by\_browser = GROUP streamed\_data by browserid;*

*by\_browser\_count = FOREACH by\_browser GENERATE FLATTEN(group) as browserid, COUNT($1);*

*sorted\_browser\_count = ORDER by\_browser\_count BY $1 DESC;*

*DUMP sorted\_browser\_count;*

The output of the aggregation command:



Pig took 1 minute and 20 seconds just to aggregate the data alone. Cleansing time is not included.

# Apache Spark

## 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
6. Start spark-master, spark-slaves and history server
7. Start spark shell with Scala with the command: spark-shell –master spark:<ip>:<port>
8. Or start spark shell with Python with the command: pyspark –master spark:<ip>:<port>

Apache Spark shell 2 variants namely Scala shell or Python Shell. The syntax for each of them varied slightly. It is mentioned that Spark with Scala is more efficient than Python.

## Spark Shell (Scala)

### Cleansing data

*import org.apache.spark.sql.functions.\_*

*val df2 = df.withColumn("browserid", when(col("browserid") === "Firefox", "Mozilla Firefox").otherwise(col("browserid")))*

*val df3 = df2.withColumn("browserid", when(col("browserid") === "Mozilla", "Mozilla Firefox").otherwise(col("browserid")))*

*val df4 = df3.withColumn("browserid", when(col("browserid") === "InternetExplorer", "Internet Explorer").otherwise(col("browserid")))*

*val df5 = df4.withColumn("browserid", when(col("browserid") === "IE", "Internet Explorer").otherwise(col("browserid")))*

*val df6 = df5.withColumn("browserid", when(col("browserid") === "Chrome", "Google Chrome").otherwise(col("browserid")))*

*df6.write.csv("hdfs:///user/dante/scala-cleansed")*

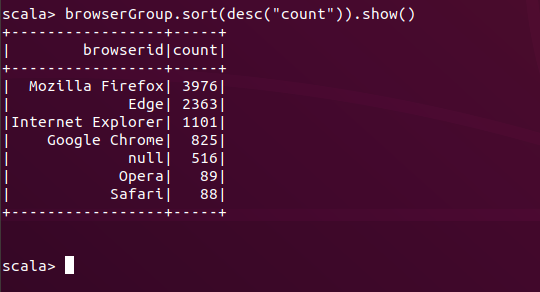
The cleansed data is stored into HDFS

### Aggregation

*val df = spark.read.format("csv").option("header","true").load("hdfs:///user/dante/clickdata\_header.csv")*

*val browserGroup = df.groupBy("browserid").count()*

*browserGroup.sort(desc("count")).show()*



## PySpark (Python)

### Cleansing

*from pyspark.sql.functions import when*

*df2 = df.withColumn("browserid", when (df['browserid'] == 'Firefox', 'Mozilla Firefox').otherwise(df['browserid']))*

*df2 = df2.withColumn('browserid', when (df2['browserid'] == 'Mozilla', 'Mozilla Firefox').otherwise(df2['browserid']))*

*df2 = df2.withColumn('browserid', when (df2['browserid'] == 'InternetExplorer', 'Internet Explorer').otherwise(df2['browserid']))*

*df2 = df2.withColumn('browserid', when (df2['browserid'] == 'IE', 'Internet Explorer').otherwise(df2['browserid']))*

*df2 = df2.withColumn('browserid', when (df2['browserid'] == 'Chrome', 'Google Chrome').otherwise(df2['browserid']))*

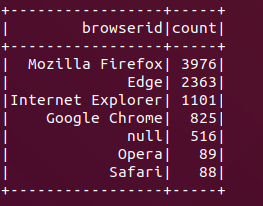
*df2.write.csv('hdfs:////user/dante/spark-cleansed')*

### Aggregation

*df = spark.read.format("csv").option("header", "true").load("hdfs:////user/dante/clickdata\_header.csv")*

*browserGroup = df.groupBy("browserid").count()*

*browserGroup.orderBy('count', ascending=False).show()*



## 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:

