**MSc in Data Analytics (SB+) - Sept 2023 - 2024 - YR1**

Author: Jose Maria Rico Leal

e-mail: sba23021@student.cct.ie

Student ID: sba23021

GitHub Link:

ca2 twitter dataset big data and data visualization

## **Abstract**

*This report focuses on processing and analysing a Twitter dataset, to process the data technologies like Hadoop, MapReduce, MySQL, Cassandra are demonstrated also an exhaustive database performance analysis is performed using Yahoo Cloud Serving Benchmark. To analyse the dataset a neural network using time series is applied to detect and predict any sentiment change throughout the given period….*

[**Abstract** 1](#_Toc165216309)

[**1.** **Introduction** 3](#_Toc165216310)

[**2.** **Big Data** 3](#_Toc165216311)

[2.1. Data storage and processing activities 3](#_Toc165216312)

[2.1.1. Storing *ProjectTweets.csv* in HDFS 4](#_Toc165216313)

[2.1.2. *MapReduce* jobs 4](#_Toc165216314)

[2.1.2.1. Count and display mentions and hashtags 4](#_Toc165216315)

[2.1.2.2. Distinct *tweet\_Id* count 5](#_Toc165216316)

[2.1.2.3. Cleaning tweets removing commas and quotes 7](#_Toc165216317)

[2.1.2.4. Ordering dataset based on *ids* 7](#_Toc165216318)

[2.1.3. Spark 9](#_Toc165216319)

[2.1.4. Hive 9](#_Toc165216320)

[2.1.5. *MySQL* 10](#_Toc165216321)

[2.1.6. Cassandra 11](#_Toc165216322)

[2.2. Database comparison for *Cassandra, MySQL* and *MongoDB* using YCSB. 11](#_Toc165216323)

[2.2.1. Workload A: Update heavy 12](#_Toc165216324)

[2.2.2. Workload B: Read mostly 12](#_Toc165216325)

[2.2.3. Workload C: Ready only 12](#_Toc165216326)

[2.2.4. Workload D: Read latest 12](#_Toc165216327)

[2.2.5. Workload E: Short ranges 12](#_Toc165216328)

[2.2.6. Workload F: Read-modify-write 12](#_Toc165216329)

[2.2.7. Workload G: Write only 12](#_Toc165216330)

[2.2.8. Rows load: inserts 12](#_Toc165216331)

[2.3. Rationale and justification for data processing, storage, and programming language 12](#_Toc165216332)

[2.4. Big Data diagram 12](#_Toc165216333)

[**3.** **Advanced Data Analytics** 12](#_Toc165216334)

[3.1. DA 12](#_Toc165216335)

[**4.** **Conclusion** 12](#_Toc165216336)

[**5.** **References** 12](#_Toc165216337)

[**6.** **Annex** 13](#_Toc165216338)

[6.1. Big Data 13](#_Toc165216339)

[6.1.1. Storing *ProjectTweets.csv* in HDFS 13](#_Toc165216340)

[6.1.2. *MapReduce* count and display mentions and hashtags 14](#_Toc165216341)

[6.1.3. *MapReduce* distinct *tweet\_id* count 16](#_Toc165216342)

[6.1.4. *MapReduce* cleaning tweets removing commas and quotes 16](#_Toc165216343)

[6.1.5. *MapReduce* ordering dataset based on *ids* 17](#_Toc165216344)

[6.1.6. *Spark* 19](#_Toc165216345)

[6.1.7. *Hive* 19](#_Toc165216346)

[6.1.8. *MySQL* 21](#_Toc165216347)

[6.1.9. *Cassandra* 22](#_Toc165216348)

[6.1.10. *YCSB: MySQL* 23](#_Toc165216349)

[6.1.11. *YCSB: MongoDB* 24](#_Toc165216350)

[6.1.12. *YCSB: Cassandra* 24](#_Toc165216351)

# **Introduction**

This project is divided in two sections Big Data and Advanced Data Analytics.

For Big Data a csv file is processed using Hadoop, from there *MapReduce* is applied to perform four tasks. Reading data from Hadoop is achieved using Spark for streaming and Hive for badging. As a non *HDFS* databases *MySQL* and Cassandra are also explored to finalise this section a thorough analysis for MySQL, Cassandra and MongoDB is carried out using Yahoo Cloud Serving Benchmark (YCSB).

The second section Advanced Data Analytics….

# **Big Data**

# Data storage and processing activities

According to Manwal and Gupta (2017), large organizations such as *Twitter, Facebook*, and *LinkedIn* use Hadoop to handle the vast amounts of data they generate daily. As the starting point of this project is the dataset *ProjectTweets.csv*, it would be beneficial to emulate the data processing methods used in *Twitter* analytics department.

# Storing *ProjectTweets.csv* in HDFS

Full implementation of this step can be seen at the annex section 6.1.1, the relevant part is that file now is into Hadoop and from there *MapReduce* jobs can be deployed, data can be read using Hive and data can be streamed for the analytics part using Spark.

A screenshot of a computer

Description automatically generated  
Fig. 1. *ProjectTweets.csv* stored into Hadoop.

# *MapReduce* jobs

Four different *MapReduce* jobs have been implemented with the aim of demonstrating how to perform these tasks. These jobs were also necessary to identify duplicates within ProjectTweets.csv, to clean the dataset before importing it into *Cassandra* and *MySQL* (as the last column contains commas and quotes, which are incompatible with those databases), and to demonstrate that Hive can achieve the same outcomes as a *MapReduce* job. Please note that full *MapReduce* jobs implementation can be found in the annex sections 6.1.2, 6.1.3, 6.1.4 and 6.1.5.

# Count and display mentions and hashtags

A given task could be to count all mentions and hashtags contained in this file. The mapper processes tweet text to find hashtags and mentions and then emits them as intermediate key-value pairs where the keys are the entities with a prefix, values are all 1 indicating a single occurrence for each entity.

A computer screen shot of a program code

Description automatically generated  
Fig. 2. *mapper.py*

The reducer sums above calculated occurrences to get a count of how often each hashtag and mention appears.

A screenshot of a computer program

Description automatically generated  
Fig. 3. *reducer.py*

# Distinct *tweet\_Id* count

This *MapReduce* job is paired with a Hive query that will produce the same output, counting distinct values for *tweet\_id*. This mapper outputs each *tweet\_id* it encounters as a key-value pair, with the *tweet\_id* as the key and 1 as the value.

A screenshot of a computer screen

Description automatically generated  
Fig. 4. *mapperd.py*

The reducer counts unique *tweet\_id* values received from the mapper.

A screenshot of a computer program

Description automatically generated  
Fig. 5. *reducerd.py*

Result to be compared with Hive query, from 1,600,000 rows, 1,598,315 are unique *tweet\_id*.

  
Fig. 6. distinct *tweet­\_id* count

# Cleaning tweets removing commas and quotes

After several attempts to insert *ProjectTweets.csv* into *Cassandra* and *MySQL*, it was impossible because the text in the last column was full of commas (","). Since a comma is used as a delimiter, every attempt to import disrupted the file structure, which did not match the created table structure. Hence, this *MapReduce* job was necessary. The mapper reads all the lines, stripping any commas and quotes. It preserves only the first five commas to delimit six columns, ensuring the structure matches the table for a smooth load.

A screenshot of a computer program

Description automatically generated  
Fig. 7. *mapperc.py*

The reducer merely passes the cleaned data through and saves it.

A screenshot of a computer

Description automatically generated  
Fig. 8. *reducerc.py*

# Ordering dataset based on *ids*

Above *MapReduce* output was not ordered by *ids. MapReduce* paradigm does not guarantee an ordered output.

A screenshot of a computer screen

Description automatically generated  
Fig. 9. Cleaning tweets removing commas and quotes output.

This mapper transforms raw input into a structured key-value format, separating *ids* from the rest of the data.

A screenshot of a computer

Description automatically generated  
Fig. 10. *mappero.py*

The reducer sorts the output by the key *ids.*

A computer screen shot of a computer code

Description automatically generated  
Fig. 11. *reducero.py*

Having the data ordered is useful for *MySQL*, as this database inserts the data in the given order. In contrast, Cassandra, like *MapReduce*, does not input the data in an ordered manner due to its distributed nature. Both *MapReduce* and *Cassandra* are designed to handle large-scale data across distributed systems, which prioritize scalability and fault tolerance over maintaining data order. This distribution means that data is processed in parallel across multiple nodes, making the preservation of order less practical and often unnecessary for the intended analytical or transactional operations.

A screenshot of a computer screen

Description automatically generated  
Fig. 12. *MapReduce* output dataset ordered by *ids.*

# Spark

# Hive

*Facebook* engineers developed this technology in 2010 to simplify the complexity of writing *MapReduc*e jobs by utilizing *SQL* syntax. *Facebook's* analysts were familiar with *SQL,* which is why this querying language was used to extract information from its vast *Hadoop* datasets (Thusoo et al., 2010). Establishing an analogy the complexity of point *2.1.2.2. Distinct tweet\_Id count* can be solved in just one line of code, *SELECT COUNT(DISTINCT tweet\_id) FROM tweets;*

A screenshot of a computer

Description automatically generated

Fig. X. *Hive* output.

Both *MapReduce* and *Hive* yield the same outcome of 1,598,315 distinct t*weet\_ids.* This is a simple and practical way of demonstrating why *Hive* was developed.

# *MySQL*

In section 2.2 a comparative database analysis for *MySQL* will be conducted. Prior to that, the output of the fourth *MapReduce* job (a dataset ordered by ID) was smoothly introduced into MySQL.

*A computer screen shot of a keyboard

Description automatically generated*Fig. X. *MySQL* queries.

One of the strong points of *MySQL* is its syntax, which is easy to interpret and perform.

A screenshot of a computer

Description automatically generated  
Fig. X. *MySQL* query.

# Cassandra

Also, *Cassandra* will be evaluated in Section 2.2. Before that, the output from *MapReduce* job four was loaded. It is worth mentioning in this section the problems I encountered before concluding that *Cassandra* does not like commas prior to a data load; it was also skipping rows that contained quotes. In the screenshot below, the rows are not ordered by *ids*. This is because *Cassandra*, due to its distributed nature, does not concern itself with order but simply distributes the data across its nodes.

**A screenshot of a computer

Description automatically generated**  
Fig. X. *Cassandra* query first 20 rows.

# Database comparison for *Cassandra, MySQL* and *MongoDB* using YCSB.

According to Manwal and Gupta (2017), large organizations such as Twitter, Facebook, and LinkedIn use Hadoop to handle the vast amounts of data they generate daily. As the starting point of this project is the dataset ProjectTweets.csv, it would be beneficial to emulate the data processing methods used in Twitter analytics department.

# Workload A: Update heavy

# Workload B: Read mostly

# Workload C: Ready only

# Workload D: Read latest

# Workload E: Short ranges

# Workload F: Read-modify-write

# Workload G: Write only

# Rows load: inserts

# Rationale and justification for data processing, storage, and programming language

# Big Data diagram

# **Advanced Data Analytics**

# DA

# **Conclusion**

# **References**

Manwal, M. and Gupta, A. (2017). 'Big data and hadoop — A technological survey', 2017 *International Conference on Emerging Trends in Computing and Communication Technologies (ICETCCT)*. DOI: 10.1109/ICETCCT.2017.8280345.

Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., Murthy, R. and Liu, H. (2010). Data warehousing and analytics infrastructure at Facebook. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. doi: 10.1145/1807167.1807278.

# **Annex**

# Big Data

# Storing *ProjectTweets.csv* in HDFS

Initialize Hadoop and make sure is running:  
$ start-dfs.sh  
$ start-yarn.sh  
$ jps

A screenshot of a computer

Description automatically generated

Locate *ProjectTweets.csv* usually it should be in Downloads:

A screenshot of a computer

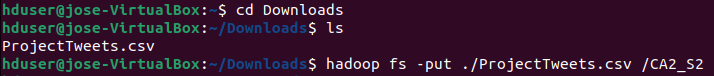
Description automatically generated

Before moving it into Hadoop a new directory for this file to be moved in must be created:  
$ Hadoop fs -mkdir /CA2\_S2  
$ Hadoop fs -ls /

A screenshot of a computer

Description automatically generated

Moving the file into Hadoop:  
$ hadoop fs -put ./ProjectTweets.csv /CA2\_S2



Listing the file into Hadoop directory CA2\_S2:  
$ hadoop fs -ls /CA2\_S2

A computer screen shot of numbers and letters

Description automatically generated

# *MapReduce* count and display mentions and hashtags

Before starting the mapper and reducer make sure permissions are granted:  
$ chmod 700 mapper.py  
$ chmod 700 reducer.py

Start the mapper.py and the reducer.py by typing this command:  
$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -mapper ./mapper.py -reducer ./reducer.py -input /CA2\_S2/ProjectTweets.csv -output /CA2\_S2\_mr1

A screenshot of a computer

Description automatically generated

Output result it is printed in Hadoop directory CA2\_S2\_mr1:  
$ hadoop fs -cat /CA2\_S2\_mr1/part-00000

A computer screen shot of a computer

Description automatically generated

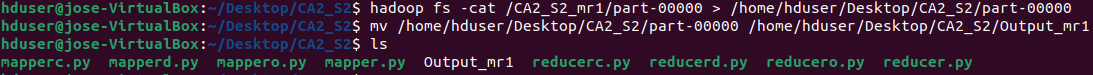
A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Move output to your local directory for readability:  
$ hadoop fs -cat /CA2\_S2\_mr1/part-00000 > /home/hduser/Desktop/CA2\_S2/part-00000  
 $ mv /home/hduser/Desktop/CA2\_S2/part-00000 /home/hduser/Desktop/CA2\_S2/Output\_mr1



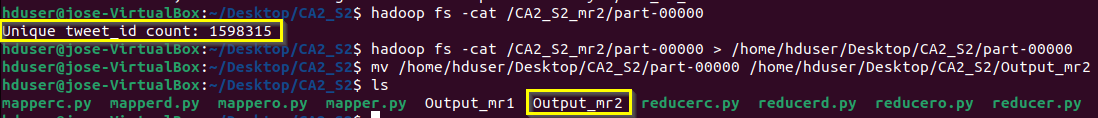
# *MapReduce* distinct *tweet\_id* count

Start *MapReduce* by typing the following command:  
$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -mapper ./mapperd.py -reducer ./reducerd.py -input /CA2\_S2/ProjectTweets.csv -output /CA2\_S2\_mr2

A screenshot of a computer program

Description automatically generated

Displaying *MapReduce* result, moving it to a local folder and renaming it:  
$ hadoop fs -cat /CA2\_S2\_mr2/part-00000  
$ hadoop fs -cat /CA2\_S2\_mr2/part-00000 > /home/hduser/Desktop/CA2\_S2/part-00000re  
$ mv /home/hduser/Desktop/CA2\_S2/part-00000 /home/hduser/Desktop/CA2\_S2/Output\_mr2



# *MapReduce* cleaning tweets removing commas and quotes

Start *MapReduce* by typing the following command:  
$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -mapper ./mapperc.py -reducer ./reducerc.py -input /CA2\_S2/ProjectTweets.csv -output /CA2\_S2\_mr3

A screenshot of a computer

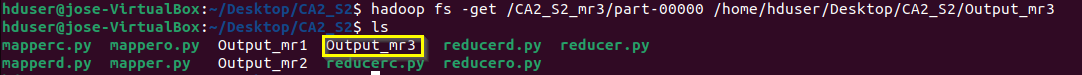
Description automatically generated

Displaying 20 rows to see if last column *text* has no commas and quotes:  
$ hadoop fs -cat /CA2\_S2\_mr3/part-00000 | head -n 20

A screenshot of a computer screen

Description automatically generated

Moving output to local folder and renaming it:  
$ hadoop fs -get /CA2\_S2\_mr3/part-00000 /home/hduser/Desktop/CA2\_S2/Output\_mr3

****

# *MapReduce* ordering dataset based on *ids*

Start *MapReduce* by typing the following command:  
$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \

-D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \  
 -D stream.num.map.output.key.fields=2 \  
 -D mapreduce.map.output.key.field.separator=, \  
 -D mapreduce.partition.keycomparator.options="-k1,1n" \  
 -files ./mappero.py,./reducero.py \  
 -mapper mappero.py \  
 -reducer reducero.py \  
 -input /CA2\_S2\_mr3/part-00000 \  
 -output /CA2\_S2\_mr4

A screenshot of a computer program

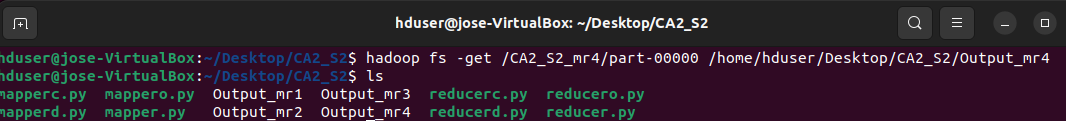
Description automatically generated

Displaying 20 rows to see if rows are displaying in ascending order:  
$ hadoop fs -cat /CA2\_S2\_mr4/part-00000 | head -n 20

A screenshot of a computer screen

Description automatically generated

Moving output to local folder and renaming it:  
$ hadoop fs -get /CA2\_S2\_mr4/part-00000 /home/hduser/Desktop/CA2\_S2/Output\_mr4



# *Spark*

Locate ProjectTweets.csv usually it should in Downloads:

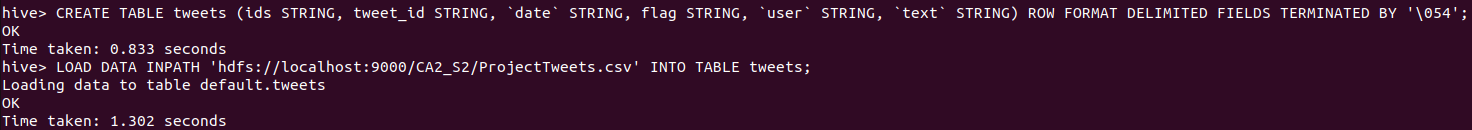
# *Hive*

To run *Hive* first start *Hadoop* and right after activate *Hive* bash:  
$ start-dfs.sh  
$ start-yarn.sh  
$ cd /usr/local/hive/bin  
$ hive

A screenshot of a computer screen

Description automatically generated

Table creation *tweets* and data load:  
CREATE TABLE tweets (ids STRING, tweet\_id STRING, `date` STRING, flag STRING, `user` STRING, `text` STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054';  
LOAD DATA INPATH 'hdfs://localhost:9000/CA2\_S2/ProjectTweets.csv' INTO TABLE tweets;



Select first 10 rows to see table structure and also make sure that the 1,600,000 went through:  
select \* from tweets limit 10;  
select count(\*) from tweets;

A computer screen shot of a person

Description automatically generated

Counting distinct rows from *tweet\_id:*  
SELECT COUNT(DISTINCT tweet\_id) FROM tweets;

A screenshot of a computer

Description automatically generated

# *MySQL*

Connect to *MySQL,* create *twitterdb,* use this database, create table *tweets,* load the data and retrieve fist 10 rows:  
create database twitterdb;  
use twitterdb;  
CREATE TABLE tweets (ids text,tweet\_id text,`date` VARCHAR(255),flag VARCHAR(255),`user` VARCHAR(255),`text` text);  
LOAD DATA INFILE '/var/lib/mysql-files/Output\_mr4' INTO TABLE tweets FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' (ids, tweet\_id, date, flag, user, text);

A computer screen shot of a computer error

Description automatically generated

Before loading the data, *Output\_mr4* has to be copied into *mysql-files:*sudo cp /home/hduser/Desktop/CA2\_S2/Output\_mr4 /var/lib/mysql-files/

A black and white text

Description automatically generated

# *Cassandra*

To initialize it insert:  
$ cd /usr/local/cassandra/  
$ bin/cassandra -f

Right after in a different terminal type:  
$ cd /usr/local/cassandra/  
$ bin/cqlsh

A screenshot of a computer screen

Description automatically generated

Once in *Cassandra* bash type the following to insert *MapReduce* four output:  
DESCRIBE KEYSPACES;  
CREATE KEYSPACE twitterdb WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };  
use twitterdb;  
CREATE TABLE tweets(ids text, tweet\_id text, date text, flag text, user text, text text, primary key (ids, tweet\_id, date, flag, user, text));  
COPY tweets FROM '/home/hduser/Desktop/CA2\_S2/Output\_mr4' WITH HEADER = FALSE AND DELIMITER = ',';

A screenshot of a computer program

Description automatically generated

Display first 20 rows:  
SELECT \* FROM tweets LIMIT 20;

**A screenshot of a computer

Description automatically generated**

# *YCSB: MySQL*

Fist we need to create a database and a table in order to load rows and perform the tests, start your mysql:  
mysql -u root -p  
create database BenchTest;  
use BenchTest;  
CREATE TABLE usertable (YCSB\_KEY VARCHAR(255) PRIMARY KEY,  
FIELD0 VARCHAR(255), FIELD1 VARCHAR(255),  
FIELD2 VARCHAR(255), FIELD3 VARCHAR(255),  
FIELD4 VARCHAR(255), FIELD5 VARCHAR(255),  
FIELD6 VARCHAR(255), FIELD7 VARCHAR(255),  
FIELD8 VARCHAR(255), FIELD9 VARCHAR(255));

A screenshot of a computer

Description automatically generated

Open a new terminal and go to *ycsb* directory, display all workloads:  
$ cd /home/hduser/ycsb-0.17.0  
$ cd workloads  
$ ls

A screenshot of a computer screen

Description automatically generated

Before running the workloads, ensure that the fields *recordcount* and *operationcount* have the same number. To carry out the tests, five iterations will be done, changing these values from 1,000 to 10,000, 50,000, 100,000, and 200,000. Ultimately, for each iteration, these rows will be inserted into MySQL. Remember that between iterations, lines should be deleted; otherwise, new lines cannot be inserted. Let us see the steps:  
$ nano workloada  
$ nano workloadb  
$ nano workloadc  
$ nano workloadd  
$ nano workloade  
$ nano workloadf  
$ nano workloadg

A screenshot of a computer screen

Description automatically generated

Remember for each iteration to make sure that lines match:

A screenshot of a computer

Description automatically generated

A table with numbers and a few words

Description automatically generated

Once workloads are adjusted, rows must be loaded into previously created table *username* by running the following commands:  
$ cd ..  
$ ./bin/ycsb.sh load jdbc -P ./jdbc-binding/conf/db.properties -P workloads/workloada

A screenshot of a computer program

Description automatically generated

Now go back to *MySQL* and see what *ycsb* has inserted into *usertable:*

A screen shot of a computer screen

Description automatically generated

Random characters allocated for testing—1,000 set in Workload A—are inserted; now we can start running the workloads. For practicality, I will be running Workload A next. The subsequent workloads are the same; please adjust accordingly:  
./bin/ycsb.sh run jdbc -P workloads/workloada -P ./jdbc-binding/conf/db.properties

A screenshot of a computer program

Description automatically generated

The highlighted output is key to comparing workload performance across databases. Ensure you copy it to plot later. The rest of the commands for the workloads are as follows:  
./bin/ycsb.sh run jdbc -P workloads/workloadb -P ./jdbc-binding/conf/db.properties  
./bin/ycsb.sh run jdbc -P workloads/workloadc -P ./jdbc-binding/conf/db.properties  
./bin/ycsb.sh run jdbc -P workloads/workloadd -P ./jdbc-binding/conf/db.properties  
./bin/ycsb.sh run jdbc -P workloads/workloade -P ./jdbc-binding/conf/db.properties  
./bin/ycsb.sh run jdbc -P workloads/workloadf -P ./jdbc-binding/conf/db.properties  
./bin/ycsb.sh run jdbc -P workloads/workloadg -P ./jdbc-binding/conf/db.properties

# *YCSB: MongoDB*

# *YCSB: Cassandra*