

CCT College Dublin

Assessment Cover Page

Module Title:	Storage Solutions for Big Data
Assessment Title:	CA2 Individual
Lecturer Name:	Dr. Muhammad Iqbal
Student Full Name:	Simone Finelli
Student Number:	sba22524
Assessment Due Date:	21/05/2023
Date of Submission:	20/05/2023

Declaration

By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

Question 1

a) Can you define Big Data? Explain major characteristics of Big Data. Can banks enhance their profits with the support of big data processing and analysis? Research and name top three businesses that have obtained the benefits of big data storage solutions in the recent past.

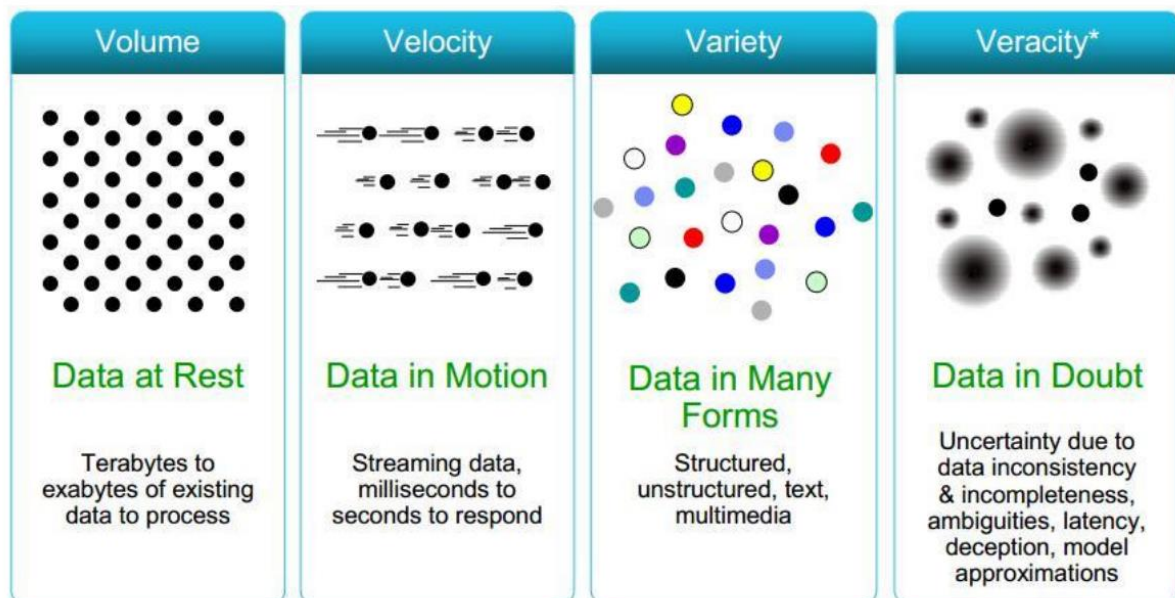
In the current world, the term Big Data is used to describe the **vast amount of structured and unstructured data** that is **complex to be processed** using traditional RDBMS and data processing applications.

This data is usually generated from different sources where challenges such as ingestion, storage, search, analysis, prediction, and visualization are encountered.

Big Data is better **described** and **characterized** by the following Vs:

- **Volume** – Big data refers to very large data sets that are too big to be processed using traditional computing systems.
- **Velocity** – Big data is generated at an unprecedented speed, making it difficult to store, process, and analyse.
- **Variety** – Big data includes a wide range of data types, including structured, semi-structured, and unstructured data from various sources.

In some cases, we can also use a 4th V – Veracity – to further explain how Big Data is often unreliable and may contain errors, making it difficult to extract meaningful insights.



4 Vs of Big Data. Source: "Lecture 1 - Introduction to SSBD"

It is very important to note that to solve Big Data problems a proper architecture such as a distributed system is needed. Expected **features** are:

- **Resource sharing** – The ability to have multiple resources, such as compute, distributed in different locations but able to cooperate for a final goal.
- **Openness** – The ability to be flexible, meaning that the system can be extended with new components.
- **Concurrency** – The ability to accomplish work in parallel, meaning that each resource in the distributed system can operate without having to wait for other processes to complete.
- **Scalability** – The ability to handle increase of work by adding more resources.
- **Fault tolerance** – The ability to continue the work even in case of failure of some components.
- **Transparency** – The ability to work with the data even if it is in multiple locations and always have visibility at which step of the collection, processing, and analysis we are at.

Banks can enhance their profits thanks to Big Data by processing and analysing large volumes of customer data, gaining insights into customer behaviour, preferences, and needs. Example of use cases are:

- **Personalized solutions for customers** – by looking at customers behaviour and habits, banks can build tailored offers while preventing churn.
- **Customer segmentation** – by clustering customer profiles, banks can build appropriate marketing strategies to acquire new clients.
- **Fraud detection** – by analysing customers behaviours, it become easier to identify anomalous transactions.
- **Lending decisions** – touching the core business of a bank, advanced analytics can support in the decision of lending a loan that must be paid off.

Source: Mathur, V. (2022)

Some **top business making advantage of Big Data** are:

- **American Express**

“A key cost for American Express is covering for fraudulent transactions that take place through its payments network or by the use of its cards. The company leverages the vast amounts of data it collects from cardholders and merchants to make fraud assessments in fractions of a second, making sure that the purchasing experience is seamless for legitimate customers while limiting the number of fraudulent transactions that are approved. For example, American Express leverages cardholder membership information, spending trends, merchant details to triangulate whether card-not-present transactions (mostly ecommerce-related) are legitimate or not. In less than a second, American Express’ ML algorithms and fraud prevention tools analyze thousands of datapoints of merchant and cardholder alike to minimize the risk of fraud.”

Source: Llano, M. (2022)

- **Facebook**

"A new generation of applications has arisen at Facebook that require very high write throughput and cheap and elastic storage, while simultaneously requiring low latency and disk efficient sequential and random read performance. MySQL storage engines are proven and have very good random read performance, but typically suffer from low random write throughput. It is difficult to scale up our MySQL clusters rapidly while maintaining good load balancing and high uptime. Administration of MySQL clusters requires a relatively high management overhead and they typically use more expensive hardware. Given our high confidence in the reliability and scalability of HDFS, we began to explore Hadoop and HBase for such applications."

Source: Meta Research. (2011)

- **AirBnB**

"Airbnb's price suggestion engine, which took months to develop and pulls on five billion training data points, has two main components: modeling and machine learning, explained Airbnb data scientist Bar Ifrach at Thursday's conference. The model pulls together what Airbnb's huge data set can reveal about a listing's best price based on things like its neighborhood and the size of the listing."

Source: Huet, E. (2015)

b) Think about a scenario where a business is having trouble meeting client needs because standard relational database systems are taking too long to process data. Describe how Big Data solutions can allow the firm to quickly gain insight for real-time data analytics. Give a brief understanding of the key actions that should be taken to deploy a Big Data framework for this firm in order to address their problems. You can assume any additional assumptions and highlight illustrations if you like to address these problems of the above-mentioned organization.

Imagine a retail company that is exponentially growing across different countries and **collecting a huge amount of data**, such as customer behaviour, store transactions and inventory.

Imagine now that that information needs to be processed to gain insights to make better decisions. Examples of analysis might be:

- Best buy product per store
- Average transaction per age
- Inventory forecasting

A **relational database can struggle** because of the following factors:

- **Data complexity** – complex queries involve the use of multiple operations and those might result in slow responses, causing decision delays.
- **Scalability** – scaling translates to hardware upgrade and cost increase, as opposed to the easier scalability of distributed systems.
- **Data variety and schema rigidity** – combining relational data to unstructured data is not feasible with traditional relational databases, this is because relational databases have schema rigidity.

Key actions for a retail company that wants to invest in Big Data might be:

- **Business understanding** – You need to first identify the problem you are trying to solve.
- **Data understanding** – You must make sure you are already collecting at scale the data that will solve the problem.
- **Technology selection** – You would define technical requirements that translates in technology shortlisting.
- **Architecture design** – You want an architecture that is reliable, scalable, and modular is crucial to address current and future desires.

A **real scenario example** is the one from **Facebook** in the previous section, where high throughput and cheap scalable storage was the reason why they decided not to use MySQL for their real-time applications.

Words for part A: 421 words

Words for part B: 255 words

Total words for question 1: 676 words

Question 2

a) What is HDFS, and how does it differ from a traditional file system? Describe and explain different layers of Hadoop framework. Explain five important characteristics of the Hadoop framework. Show the deployment of Hadoop on your virtual machine (VM) by providing the screenshots of (Namenode, Datanode etc.) and your username clearly shows your VM.

HDFS (Hadoop Distributed File System) is a **distributed and scalable file system** designed to store and manage large datasets across multiple machines. It is a key component of the Hadoop ecosystem.

Compared to a traditional file system, HDFS has several differences. Most important ones are:

- HDFS is **designed to work with Big Data**, being able to handle and process petabytes of data.
- HDFS is **fault-tolerant by design**, meaning that if a failure occurs data can be recovered. This happens thanks to the replication of data across multiple nodes.
- HDFS is **scalable horizontally**, meaning that if data grows you can just add new nodes in the cluster.

Different layers of the Hadoop framework are:

- **HDFS** – This layer is for storing data at scale, as previously described.
- **MapReduce** – This layer is the computational engine responsible for the processing of data across the cluster.
- **YARN** (Yet Another Resource Negotiator) – This layer allows the management of resources and supports the scheduling of jobs, also supporting the execution of other services in the Hadoop ecosystem.
- **Hadoop Common** – It is a collection of common libraries and utilities needed to support other the wider ecosystem.
- **Ecosystem** – It is a set of additional components that can enhance the core functionalities of Hadoop. Examples include: HBase, Spark, Hive and Pig.

Important characteristics are:

- **Scalability** – It is designed to scale horizontally, allowing customers to analyse and store very large datasets in a cluster.
- **Fault tolerance** – It is designed to be fault-tolerant, thanks to the replication of data across different nodes.
- **Flexibility** – It allows you to work with different types of data such as structured, non-structured and semi-structured.
- **Cost-effective** – It doesn't require a license as it is open-source and can run on cheap hardware.
- **High performance** – It is designed to exhibit great performance thanks to the possibility of process the data in parallel.

Below the screenshot of the Hadoop configuration in my own VM:

```
hduser@SSBD-VM-2023:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [SSBD-VM-2023]
hduser@SSBD-VM-2023:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hduser@SSBD-VM-2023:~$ jps
50337 ResourceManager
41858 SparkSubmit
50051 SecondaryNameNode
51507 Jps
49892 DataNode
49780 NameNode
50454 NodeManager
29787 JarBootstrapMain
hduser@SSBD-VM-2023:~$ Simone Finelli - sba22524
```

Screenshot of Hadoop configuration on personal VM

b) Demonstrate a comparison of MySQL and Apache Hive based on the architecture and performance. Consider a dataset and perform a query on both systems with at least 5,000 rows and at least 5 features. Show the duration of query execution by displaying screenshots obtained from a virtual machine (VM).

MySQL is an open-source relational database management system (RDBMS) that uses a **client-server architecture**. The server manages the logic of the database system including storage and data retrieval, the client is the component that allows the end-user to interact with the server to run the queries and manage the data.

On the other hand, **Apache Hive is a data warehousing tool available in the Hadoop ecosystem**. It refers to data stored in HDFS while providing a SQL-like interface (HiveQL) to allow users to query the data. Hive takes advantage of a **distributed architecture**, having the data distributed across multiple nodes and executing queries in parallel.

The main difference in the architecture is that **MySQL is centralized** while **Apache Hive is much more distributed**. In a real-case scenario you would typically use MySQL for **OLTP** (transactional) operations such as supporting a single application, while you would use Apache Hive for **OLAP** (analytics) operations where you want to work and analyse complex data to build, for example, business reports.

To benchmark the performance of the two systems an open dataset from Kaggle, recording **customer churn**, has been used. After some processing available in the “Data Preparation” notebook, a final dataset of 1000 rows and 6 columns was obtained:

```
In [6]: df.head()
```

```
Out[6]:
```

	CustomerId	CreditScore	Geography	Gender	Age	Balance
0	15634602	619	France	Female	42	0.00
1	15647311	608	Spain	Female	41	83807.86
2	15619304	502	France	Female	42	159660.80
3	15701354	699	France	Female	39	0.00
4	15737888	850	Spain	Female	43	125510.82

```
In [7]: df.shape
```

```
Out[7]: (10000, 6)
```

Resource of preprocessing available in the notebook “Data Preparation.ipynb”

After installing MySQL in the VM the CSV was imported into the database according to below screenshots.

```
mysql> CREATE DATABASE ssbd;
Query OK, 1 row affected (0.04 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| ssbd |
| sys |
+-----+
5 rows in set (0.00 sec)
```

Step 1. Creation of the “ssbd” database in MySQL

```
mysql> USE ssbd;
Database changed
mysql> CREATE TABLE customers (customer_id INT NOT NULL, credit_score INT NOT NULL, geography VARCHAR(255) NOT NULL, gender VARCHAR(255) NOT NULL, age INT NOT NULL, balance FLOAT NOT NULL, PRIMARY KEY (customer_id));
Query OK, 0 rows affected (0.13 sec)
```

Step 2. Creation of the table “customers” using the defined schema

```
mysql> LOAD DATA LOCAL INFILE "/home/hduser/Desktop/SSBD/CA2/data.csv" INTO TABLE customers FIELDS TERMINATED BY "," LINES TERMINATED BY "\n" IGNORE 1 ROWS (customer_id, credit_score, geography, gender, age, balance);
Query OK, 10000 rows affected (0.95 sec)
Records: 10000 Deleted: 0 Skipped: 0 Warnings: 0
```

Step 3. Loading of the local csv file in the table “customers”

```
mysql> SELECT * FROM customers LIMIT 10;
+-----+-----+-----+-----+-----+-----+
| customer_id | credit_score | geography | gender | age | balance |
+-----+-----+-----+-----+-----+-----+
| 15565701 | 698 | Spain | Female | 39 | 161994 |
| 15565706 | 612 | Spain | Male | 35 | 0 |
| 15565714 | 601 | France | Male | 47 | 64430.1 |
| 15565779 | 627 | Germany | Female | 30 | 57809.3 |
| 15565796 | 745 | Germany | Male | 48 | 96048.5 |
| 15565806 | 532 | France | Male | 38 | 0 |
| 15565878 | 631 | Spain | Male | 29 | 0 |
| 15565879 | 845 | France | Female | 28 | 0 |
| 15565891 | 709 | France | Male | 39 | 0 |
| 15565996 | 653 | France | Male | 44 | 0 |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

Step 4. Check of data load executing a simple query showing the first 10 rows from the table “customer”

The query I wanted to compare is the one returning the **average balance of people, based on their age**. Results were then sorted in descending order.

```
mysql> SHOW PROFILES;
+-----+-----+-----+
| Query_ID | Duration | Query |
+-----+-----+-----+
| 1 | 0.01944400 | SELECT age, AVG(balance) AS avg_balance FROM customers GROUP BY age ORDER BY age DESC |
+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

Step 5. Time execution of desired query

In Hive I replicated similar steps below described.

```
hive> CREATE TABLE IF NOT EXISTS customers (customer_id int, credit_score int, geography String, gender String, age int, balance float) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054';
OK
Time taken: 2.258 seconds
```

Step 1. Creation of the “customers” table in Hive

```
hive> LOAD DATA LOCAL INPATH "/home/hduser/Desktop/SSBD/CA2/data.csv" INTO TABLE customers;
Loading data to table default.customers
OK
Time taken: 3.153 seconds
```

Step 2. Loading of the local csv file in the table “customers”

```
hive> SELECT age, AVG(balance) AS avg_balance FROM customers GROUP BY age ORDER BY age DESC;
Query ID = hduser_20230514194938_fa6f9dcc-6f72-41f6-a8ac-36e8ce0073b2
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-05-14 19:49:40,607 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local391347963_0007
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-05-14 19:49:42,264 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_local822413312_0008
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 3002136 HDFS Write: 0 SUCCESS
Stage-Stage-2:  HDFS Read: 3002136 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
```

Step 3. Execution of the desired query

```
21      75926.8825913915
20      76038.8904296875
19      70502.5402199074
18      79169.57528409091
NULL      NULL
Time taken: 3.447 seconds, Fetched: 71 row(s)
```

Step 4. Performance results of the executed query

```

hive> INSERT OVERWRITE LOCAL DIRECTORY "/home/hduser/export" ROW FORMAT DELIMITED FIELDS TERMINATED BY "," SELECT age, AVG(balance) AS avg_balance FROM customers GROUP BY age ORDER BY age DESC;
Query ID = hduser_20230514195332_1e95f94f-f53e-4716-aa3d-86dc1d0cdf14
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (Local Hadoop)
2023-05-14 19:53:34,754 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1527915978_0009
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (Local Hadoop)
2023-05-14 19:53:36,208 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_local164788056_0010
Moving data to local directory /home/hduser/export
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 3752670 HDFS Write: 0 SUCCESS
Stage-Stage-2:  HDFS Read: 3752670 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 3.338 seconds

```

Step 5. Saving of the executed query to a local file “Hive_query_result”

From results we can see that **MySQL executed the query in 0.019444 seconds**, while **Hive took 3.447 seconds**. This is because **MySQL query optimizer performs very well on small datasets** but would struggle on very large datasets (such as analytical workloads) where Hive would perform much better.

Words for part A: 312 words

Words for part B: 304 words

Total words for question 2: 616 words

Question 3

a) Explain Apache Flink architecture and illustrate with your own conceptual diagram (Use of online/ book images is prohibited, Use draw.io to create the image). What is Apache Storm, and how does it differ from other distributed computing systems? Consider a text file comprising at least 20,000 words and write a wordcount program (Java/ Python) to count the frequency of words and related aggregation functions.

Apache Flink Architecture

Apache Flink's architecture is **designed for distributed stream processing and batch processing**. It uses a distributed setup and can be deployed in most famous distributed computing frameworks.

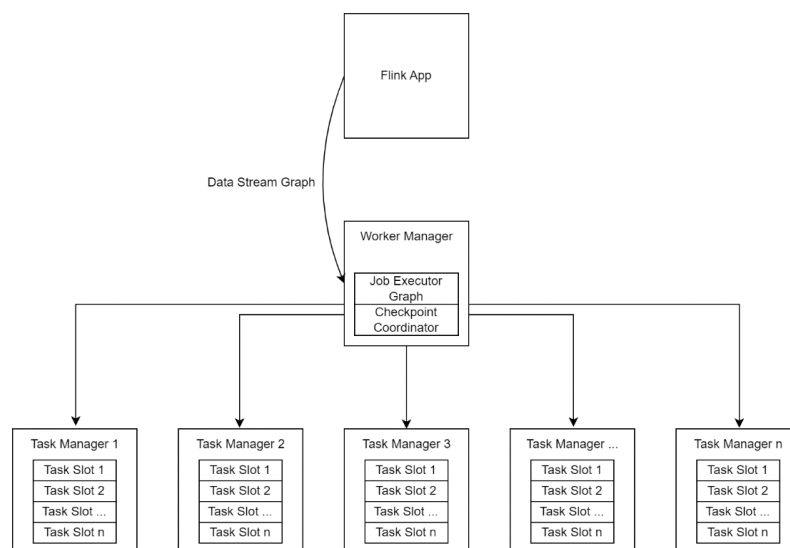
At the centre of its architecture, we can find the **master node** – the **Job Manager**. It orchestrates and manages the lifecycle of jobs submitted to Flink.

Once a job is submitted to the Job Manager, it builds a **Job Execution Graph**, which represents the data flow and transformations involved in the job. This graph represents the operations and correspondent sequence to be applied to the stream data.

Task Managers are the **slave nodes**, and they act to execute the tasks provided by the Job Manager. They execute portion of tasks called task slots. **Task slots** are the units of processing capacity available on a Task Manager. The Job Manager assigns tasks from the Job Execution Graph to the available task slots for execution. Task Managers can communicate with the Job Manager to coordinate and share results from executed tasks.

Flink architecture is **fault tolerant**, meaning that it can handle failure recovery in case of disaster to ensure reliable and resilient processing. Fault tolerance is accomplished using **checkpoint and save points**. Those are snapshot of the application state that are written to a remote and durable location.

Below a diagram of the architecture explained:



Apahce Flink architecture, also available in "Flink Architecture.drawio"

Apache Storm

Apache Storm is an **open-source distributed real-time stream processing system** that can process a million of tuples per second per node. Storm is designed for **high-throughput, low-latency, and fault-tolerant processing of streaming data** and has great capabilities in terms of **query parallelization**.

Storm is **different from other distributed computing systems**, such as Hadoop, because of:

- **Distributed Remote Procedure Call (RPC)** – Storm can parallelise very complex functions very fast to accomplish real-time computation.
- **Continuous computation** – Storm is designed to process data continuously. It means that data is processed as it comes until its topology is eventually stopped.
- **Stream processing** – Storm not only is able to process incoming stream of data from different sources, but it is also able to update different databases in real time.
- **Multiple programming languages** – Storm was designed to be accessible with any programming language thanks to the possibility to use its APIs.

To solve the requested problem, a topology has been built using the script contained in the “WordCountTopology.java” file.

```
hudson@5580-VN-2021: /bigdata/WordCountFile$ mvn clean install
[INFO] Scanning for projects...
[INFO] -----< admincloud:storm-example >-----
[INFO] Building storm-example 1.0
[INFO] -----[ jar ]-----
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ storm-example ---
[INFO] Deleting /home/hudson/bigdata/WordCountFile/target
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ storm-example ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/hudson/bigdata/WordCountFile/src/main/resources
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ storm-example ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /home/hudson/bigdata/WordCountFile/target/classes
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ storm-example ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/hudson/bigdata/WordCountFile/src/test/resources
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ storm-example ---
[INFO] No sources to compile
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ storm-example ---
[INFO] No tests to run.
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ storm-example ---
[INFO] Building jar: /home/hudson/bigdata/WordCountFile/target/storm-example-1.0.jar
[INFO] --- maven-assembly-plugin:2.2-beta-5:single (make-assembly) @ storm-example ---
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] org/ already added, skipping
[INFO] org/slf4j/ already added, skipping
[INFO] org/slf4j/helpers/ already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] org/ already added, skipping
[INFO] org/apache/ already added, skipping
[INFO] org/apache/logging/ already added, skipping
[INFO] META-INF/DEPENDENCIES already added, skipping
[INFO] META-INF/LICENSE already added, skipping
[INFO] META-INF/NOTICE already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/maven/org.apache.logging.log4j/ already added, skipping
[INFO] Building jar: /home/hudson/bigdata/WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] org/ already added, skipping
[INFO] org/slf4j/ already added, skipping
[INFO] org/slf4j/helpers/ already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] org/ already added, skipping
[INFO] org/apache/ already added, skipping
[INFO] org/apache/logging/ already added, skipping
[INFO] META-INF/DEPENDENCIES already added, skipping
[INFO] META-INF/LICENSE already added, skipping
[INFO] META-INF/NOTICE already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/maven/org.apache.logging.log4j/ already added, skipping
[INFO] --- maven-install-plugin:2.4:install (default-install) @ storm-example ---
[INFO] Installing /home/hudson/bigdata/WordCountFile/target/storm-example-1.0.jar to /home/hudson/.m2/repository/admincloud/storm-example/1.0/storm-example-1.0.jar
[INFO] Installing /home/hudson/bigdata/WordCountFile/pom.xml to /home/hudson/.m2/repository/admincloud/storm-example/1.0/storm-example-1.0.pom
[INFO] Installing /home/hudson/bigdata/WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar to /home/hudson/.m2/repository/admincloud/storm-example/1.0/storm-example-1.0-jar-with-dependencies.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.622 s
[INFO] Finished at: 2023-05-15T20:55:29+02:00
[INFO] -----
```

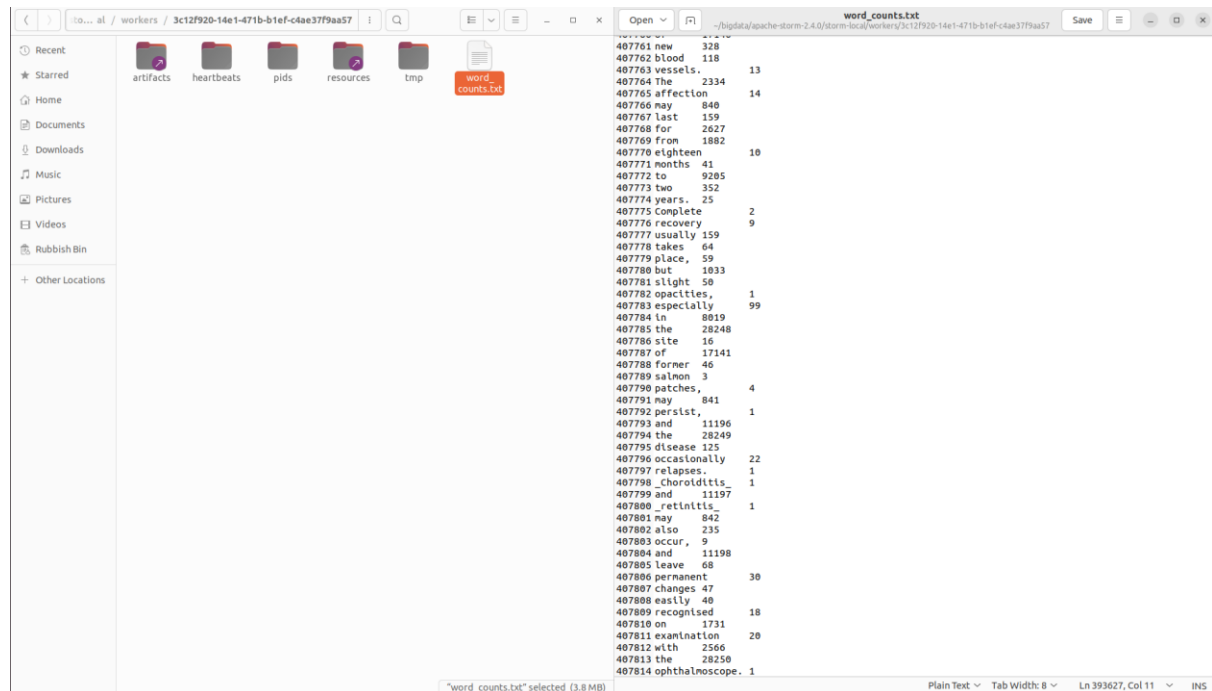
Building of “WordCountTopology.java” using maven

To execute the topology the following command was used:

```
huser@580-VN-2021: ~$ ./apache-storm-2.4.0/bin/storm jar ./WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar admicloud.storm.wordcount.WordCountTopology WordCountFile
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hduser/bigdata/apache-storm-2.4.0/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hduser/bigdata/WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Running: /usr/bin/java -Ddaemon.name= -Dstorm.options= -Dstorm.home=/home/hduser/bigdata/apache-storm-2.4.0 -Dstorm.log.dir=/home/hduser/bigdata/apache-storm-2.4.0/logs -Djava.library.path=/usr/lib64/
/usr/lib64/opt/local/lib:/usr/lib64/usr/lib64 -Dstorm.conf.file= -cp /home/hduser/bigdata/apache-storm-2.4.0/*:/home/hduser/bigdata/apache-storm-2.4.0/lib-worker/*:/home/hduser/bigdata/apache-storm-2.4.0/e
xtlib/*:/WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar:/home/hduser/bigdata/apache-storm-2.4.0/conf:/home/hduser/bigdata/apache-storm-2.4.0/bin: -Dstorm.jar=./WordCountFile/target/sto
rm-example-1.0-jar-with-dependencies.jar -Dstorm.dependency.jars= -Dstorm.dependency.artifacts={} admicloud.storm.wordcount.WordCountTopology WordCountFile
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hduser/bigdata/apache-storm-2.4.0/lib-worker/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hduser/bigdata/WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
21:15:40.142 [main] INFO o.a.s.s.StormSubmitter - Generated ZooKeeper secret payload for MD5-digest: -7738182417884461277:8456349777811557460
21:15:40.447 [main] INFO o.a.s.u.NimbusClient - Found leader nimbus : 5580-VN-2023.myguest.virtualbox.org:6027
21:15:40.454 [main] INFO o.a.s.s.a.ClientAuthUtils - Got AutoCreds {}
21:15:40.541 [main] INFO o.a.s.s.StormSubmitter - Uploading dependencies - jars...
21:15:40.543 [main] INFO o.a.s.s.StormSubmitter - Uploading dependencies - artifacts...
21:15:40.545 [main] INFO o.a.s.s.StormSubmitter - Dependency Blob keys - jars : [] / artifacts : []
21:15:40.560 [main] INFO o.a.s.s.StormSubmitter - Uploading topology jar - WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar to assigned location: /home/hduser/bigdata/apache-storm-2.4.0/st
orm-local/nimbus/inbox/stormjar-bddb1aae-8da8-46d7-be64-d801fe6f8f58.jar
Start uploading file './WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar' to '/home/hduser/bigdata/apache-storm-2.4.0/storm-local/nimbus/inbox/stormjar-bddb1aae-8da8-46d7-be64-d801fe6f8f5
8.jar' (166217 bytes)
[*****] 166217 / 166217
File './WordCountFile/target/storm-example-1.0-jar-with-dependencies.jar' uploaded to '/home/hduser/bigdata/apache-storm-2.4.0/storm-local/nimbus/inbox/stormjar-bddb1aae-8da8-46d7-be64-d801fe6f8f58.jar'
(166217 bytes)
21:15:40.674 [main] INFO o.a.s.s.StormSubmitter - Successfully uploaded topology jar to assigned location: /home/hduser/bigdata/apache-storm-2.4.0/storm-local/nimbus/inbox/stormjar-bddb1aae-8da8-46d7-be64
-d801fe6f8f58.jar
21:15:40.695 [main] INFO o.a.s.s.StormSubmitter - Submitting topology WordCountFile in distributed mode with conf {"storm.zookeeper.topology.auth.scheme":"digest","storm.zookeeper.topology.auth.payload":"
-7738182417884461277:8456349777811557460","topology.workers":3,"topology.debug":true}
21:15:40.900 [main] INFO o.a.s.s.StormSubmitter - Finished submitting topology: WordCountFile
```

Topology execution

The **processed file** was a txt named “file_text.txt” (*The Adventures of Sherlock Holmes by Sir Arthur Conan Doyle*) and the path was specified into the java program. Instead of updating any external system, I decided to print results in the file “word_counts.txt”:



The screenshot shows a file explorer window with the file 'word_counts.txt' selected. The file contains a list of words and their corresponding counts, sorted by count in descending order. The words are from the text 'The Adventures of Sherlock Holmes by Sir Arthur Conan Doyle'.

407761	new	328
407762	blood	118
407763	vessels.	13
407764	The	2334
407765	affection	14
407766	may	840
407767	Last	159
407768	for	2627
407769	from	1882
407770	eighteen	10
407771	months	41
407772	to	9205
407773	two	352
407774	years.	25
407775	complete	2
407776	recovery	9
407777	usually	159
407778	takes	64
407779	place,	59
407780	but	1033
407781	slight	50
407782	opacities,	1
407783	especially	99
407784	in	8019
407785	the	28248
407786	site	16
407787	of	17141
407788	former	46
407789	salmon	3
407790	patches,	4
407791	may	841
407792	persist,	1
407793	and	11196
407794	the	28249
407795	disease	125
407796	occasionally	22
407797	relapses.	1
407798	Choroiditis	1
407799	and	11197
407800	retinitis	1
407801	may	842
407802	also	235
407803	occur,	9
407804	and	11198
407805	leave	68
407806	permanent	30
407807	changes	47
407808	easily	40
407809	recognised	18
407810	on	1731
407811	examination	20
407812	with	2566
407813	the	28250
407814	ophthalmoscope.	1

Result of WordCountFile topology

Below screenshots of Storm environment.

← → ↺

localhost:8080

90%

🔍

Storm UI

Cluster Summary

Version	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
2.4.0	1	3	1	4	6	6

Nimbus Summary

Search:

Host	Port	Status	Version	Uptime
SSBD-VM-2023.myguest.virtualbox.org	6627	Leader	2.4.0	1d 1h 24m 28s

Showing 1 to 1 of 1 entries

Owner Summary

Search:

Owner	Total Topologies	Total Executors	Total Workers	Memory Usage (MB)
hduser	1	6	3	768

Showing 1 to 1 of 1 entries

Topology Summary

Search:

Name	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Assigned Generic Resources	Scheduler Info	Topology Version	Storm Version
WordCountFile	hduser	ACTIVE	1h 6m 53s	3	6	6	1	768	NaN			2.4.0

Showing 1 to 1 of 1 entries

Supervisor Summary

Search:

Host	Id	Uptime	Slots	Used slots	Avail slots	Used Mem (MB)	Version	Blacklisted
SSBD-VM-2023.myguest.virtualbox.org (log)	19eeaf17-e0aa-4332-ab91-e6bad300e3fe-127.0.1.1	1d 1h 22m 47s	4	3	1	0	2.4.0	false

Showing 1 to 1 of 1 entries

Storm UI view

Topology summary

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Scheduler Info	Topology Version	Storm Version
WordCountFile	WordCountFile-5-1884178146	hduser	ACTIVE	1h 7m 50s	3	6	6	1	0			2.4.0

Topology actions

Activate Deactivate Rebalance Kill Debug Stop Debug Change Log Level

Topology stats

Window	Enitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	111,050	56,352	0	0	0
3h 0m 0s	881,400	459,940	0	0	0
1d 0h 0m 0s	881,400	459,940	0	0	0
All time	881,400	459,940	0	0	0

Spouts (All time)

Search:

Id	Executors	Tasks	Enitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
spout	1	1	38,820	38,820	0.000	0	0				

Showing 1 to 1 of 1 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Enitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
count	1	1	421,460	0	0.135	1,255	421,460	0.955						
split	1	1	421,120	421,120	0.021	2,798	38,800	1,916	38,800	0				

Showing 1 to 2 of 2 entries

Worker Resources

Search: [Toggle Components](#)

Host	Supervisor Id	Port	Uptime	Num executors	Assigned Mem (MB)	Components
SSBD-VM-2023.myguest.virtualbox.org	19eeaf17-e0aa-4332-ab91-e6bad300e3fe-127.0.1.1	6701	1h 7m 4s	2	256	Components
SSBD-VM-2023.myguest.virtualbox.org	19eeaf17-e0aa-4332-ab91-e6bad300e3fe-127.0.1.1	6702	1h 7m 4s	2	256	Components
SSBD-VM-2023.myguest.virtualbox.org	19eeaf17-e0aa-4332-ab91-e6bad300e3fe-127.0.1.1	6700	1h 7m 6s	2	256	Components

WordCountFile topology view

b) Why is Apache Storm useful for Stream processing specifically? Distinguish the characteristics of Apache storm as compared to Hadoop. What is the role of Apache Zookeeper in Apache Storm deployment. Provide the screenshot of your VM to show working of Storm UI including Cluster, Nimbus and Owner summary.

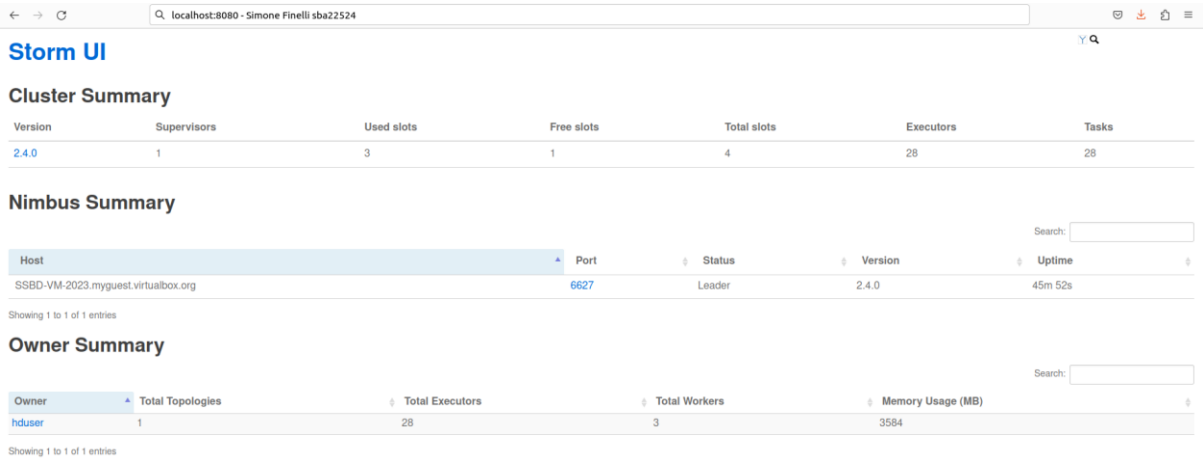
Apache Storm is useful for stream processing because it is **designed to handle large volumes of continuous data in real-time**. It processes data as it is generated, typically in small batches or individual events, rather than processing pre-stored data in large batches as in batch processing. Storm has low latency and high throughput, making it ideal for use cases such as real-time analytics, fraud detection, and IoT data processing.

Here are some key characteristics that distinguish Apache Storm from Hadoop:

- **Real-time processing** – Apache Storm is designed specifically for real-time stream processing, while Hadoop is primarily designed for batch processing of large volumes of data. Storm can process streaming data in near-real-time, typically in seconds or milliseconds.
- **Low latency and high throughput** – Because Storm is designed for real-time processing, it is optimized for low-latency and high-throughput data processing. Hadoop, on the other hand, is optimized for batch processing, which typically involves longer processing times and lower throughput.
- **Stateless** – Hadoop is stateful, meaning that intermediate states of the processing must be maintained and persisted during computation. On the other hand, Storm is stateless, in this way the internal state of previously processed and emitted data is not maintained.
- **Architecture design** – Hadoop uses HDFS to store the data and MapReduce to process it; it's worth to note that intermediate steps of the batch processing with MapReduce are stored in HDFS. Storm uses spouts as source of stream data ingestion and bolts for processing the data and send to the external output; in this case the processing occurs in-memory, meaning that it's not needed to rely on disk storage.

In a Storm deployment, **ZooKeeper** is used to **manage coordination** between the Nimbus master node and the Supervisor worker nodes. It is not used to send messages but only to coordinate the cluster management; thanks to it you can add more supervisor nodes to the cluster, and they would be made visible to the Nimbus master. Given that Storm is stateless, **some states are stored by Zookeeper**. In this way if there is a crash, work can restart where it was left.

Below the screenshot of the Storm UI working on my VM (in previous reply, more detailed screenshots when running a topology were made available):



Storm UI screenshot from personal VM

Words for part A: 438 words

Words for part B: 370 words

Total words for question 3: 808 words

Reference list

Mathur, V. (2022). Big Data In Banking Industry: Benefits, Uses and Challenges | Analytics Steps. [online] [www.analyticssteps.com](https://www.analyticssteps.com/blogs/big-data-banking-industry-benefits-uses-and-challenges). Available at: <https://www.analyticssteps.com/blogs/big-data-banking-industry-benefits-uses-and-challenges>.

Llano, M. (2022). American Express: Using Big Data to Prevent Fraud. [online] Available at: <https://d3.harvard.edu/platform-digit/submission/american-express-using-big-data-to-prevent-fraud/>.

Meta Research. (2011). Apache Hadoop goes realtime at Facebook - Meta Research. [online] Available at: <https://research.facebook.com/publications/apache-hadoop-goes-realtime-at-facebook/>.

Huet, E. (2015). How Airbnb Uses Big Data And Machine Learning To Guide Hosts To The Perfect Price. [online] Forbes. Available at: <https://www.forbes.com/sites/ellenhuet/2015/06/05/how-airbnb-uses-big-data-and-machine-learning-to-guide-hosts-to-the-perfect-price/>.

Total words visualization

