Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with 'A' Grade (CGPA 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Kautik Jolapara SAP ID: 60004200107

Course: Big Data Infrastructure Laboratory Course Code: **DJ19CEEL6011**

Year: **T.Y. B. Tech.** Sem: **VI** Batch: B1

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	EXP. 1	EXP. 2	EXP. 3	EXP. 4	EXP. 5	EXP. 6	EXP. 7	EXP. 8	EXP. 9	EXP. 10	Σ	Avg	A 1	A 2	Σ	Avg
Course Outcome	1	2	2	2	3	3	4	4	5	6						
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5	4	5	4	5	4	4						
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	4	5	4	5	4	5	4						
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	4	5	4	5	4	5	4	4	4	5						
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4	4	4	4	5	4	5						
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–	–				–	–		
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4	4	4	4	4	4	4						
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–	–	–	–	–	–				–	–		
Total	21	22	21	22	21	22	21	22	21	22						
Signature of the faculty member																

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. = 21.5	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student: Kautik

Signature of the Faculty member: Brijesh Iyer
Name of the Faculty member:

Signature of Head of the Department
Date:

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 1

Aim: Big Data Case Study with Hadoop Ecosystem.

Theory:

1. **Take examples of organization's such as Amazon, Google, Netflix etc. Flipkart.**

2. **Study the Big Data Approach they have chosen.**

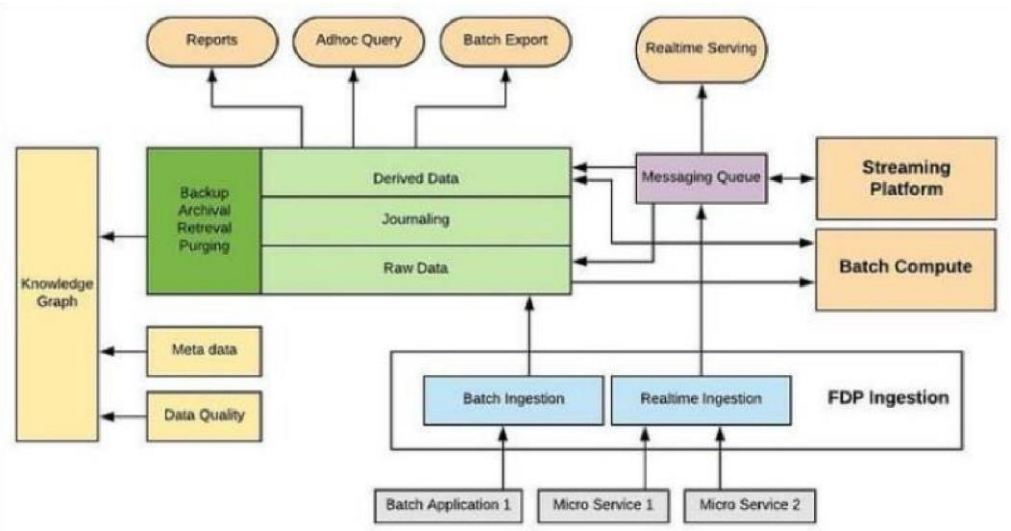
Flipkart's uses Big Data Tools and service oriented architecture (SOA) building thousands of microservices that power specific user experiences — such as search service, product listings service, pricing engine, estimating delivery dates etc. Each micro-service maintains domain data in the data store of their choice be it MySQL, HBase, Redis, Elasticsearch and more. Analytics, Insights, Data Science and even other microservice teams depend on data from multiple teams to run their business as usual. Flipkart Data Platform (FDP) provides the capabilities necessary for the teams to consume and act on this data.

FDP manages 800+ nodes Hadoop cluster to store more than 35 PB of data. They also run close to 25,000 compute pipelines on our Yarn cluster. Daily TBs of data is ingested into FDP and it also handles data spikes because of sale events. The tech stack majorly comprises of HDFS, Hive, Yarn, MR, Spark, Storm & other API services supporting the meta layer of the data.

3. **Discuss the Ecosystem Components they are using.**

Overall FDP can be broken down into following high level components.

1. Ingestion System.
2. Batch Data Processing System.
3. Real time Processing System.
4. Report Visualization.



1. Ingestion System

Each ingested payload has a fixed schema which can be created via a self serve UI. The tech stack for the system includes Messaging Queue (Kafka), Dropwizard, HDFS, Quartz, Azkaban & Hive. The user creates a schema for which a corresponding Kafka topic is created. Using Specter or Dart client the user can start ingesting the data to FDP. The ingestion system then stores the payload in HDFS files as raw Hive Tables and then journaling is run on top to make the data in the payload query able.

2. Batch Data Processing System

The ingested data which is now query able via hive queries is ready to be prepared for consumption. A user can create a Star Schema of Fact with multiple dimensions. The facts and dimensions can either be Hive tables.

3. Real time Processing System

The real time ingested data which happens via Dart or Specter clients can be directly consumed via their respective Kafka topics. Flipkart Streaming Platform (FSP) enables plugging up custom spark jobs to these topics. The streaming platform allows near real time aggregations to be built on all the ingested data.

4. Real time Processing System

Once the Analyst has created a Fact & Dimension in a columnar storage RDBMS. They can create boilerplate reports on top of the data using an in-house self serve ui. Analysts can come

and select the various metrics, filters & group by dimensions. According to the selection a set of allowed visualization charts are visible to the user.

Pros:

1. **Scalability:** Hadoop ecosystem tools like Hadoop Distributed File System (HDFS) and MapReduce are designed to scale horizontally, which means they can handle large amounts of data and can be easily expanded to meet growing data needs.

2. **Cost-effective:** Hadoop ecosystem tools are open-source, which means they are free to use and can be easily customized to meet specific business needs. This makes them an affordable solution for data management.

3. **Flexibility:** Hadoop ecosystem tools are versatile and can work with a wide range of data formats, including structured, semi-structured, and unstructured data. They can also integrate with a variety of other data processing tools and systems.

4. **Data processing:** Hadoop ecosystem tools are designed for batch processing of large amounts of data. This makes them ideal for Flipkart, which needs to process massive amounts of customer data to provide personalized recommendations, optimize supply chain management, and improve customer experience.

Cons:

1. **Complexity:** Hadoop ecosystem tools can be complex to set up and manage, requiring specialized skills and knowledge. This can add to the cost and time needed to implement and maintain the system.

2. **Performance:** Hadoop ecosystem tools are designed for batch processing, which means they may not be suitable for real-time processing of data. This can be a limitation for Flipkart, which needs to process data quickly to deliver real-time recommendations and insights to customers.

3. **Data security:** Hadoop ecosystem tools may not have the same level of security as traditional relational databases. This can be a concern for Flipkart, which needs to ensure the security of customer data.

4. **Lack of support:** Hadoop ecosystem tools are open-source, which means there is no formal support system available. This can be a challenge for Flipkart, which may need help troubleshooting issues or making updates to the system.

Conclusion:

Flipkart uses big data tools and Hadoop ecosystem tools for Scalability, Cost-Effective, Flexibility, Data processing operations though this increases the complexity it is the requirement for effective management of data. The FDP system of flipkart uses multiple components of the Hadoop ecosystem to meet the increasing demands of industry.

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 2
Installation of Hadoop on a single node cluster

AIM: Install Hadoop on a Single Node Cluster

THEORY:
Apache Hadoop is an open source framework that is used to efficiently store and process large datasets ranging in size from gigabytes to petabytes of data. Instead of using one large computer to store and process the data, Hadoop allows clustering multiple computers to analyze massive datasets in parallel more quickly.

Hadoop consists of four main modules:

- Hadoop Distributed File System (HDFS) – A distributed file system that runs on standard or low-end hardware. HDFS provides better data throughput than traditional file systems, in addition to high fault tolerance and native support of large datasets.
- Yet Another Resource Negotiator (YARN) – Manages and monitors cluster nodes and resource usage. It schedules jobs and tasks.
- MapReduce – A framework that helps programs do the parallel computation on data. The map task takes input data and converts it into a dataset that can be computed in key value pairs. The output of the map task is consumed by reduce tasks to aggregate output and provide the desired result.
- Hadoop Common – Provides common Java libraries that can be used across all modules.

How Hadoop Works

Hadoop makes it easier to use all the storage and processing capacity in cluster servers, and to execute distributed processes against huge amounts of data. Hadoop provides the building blocks on which other services and applications can be built.

Applications that collect data in various formats can place data into the Hadoop cluster by using an API operation to connect to the NameNode. The NameNode tracks the file directory structure and placement of “chunks” for each file, replicated across DataNodes. To run a job to query the data, provide a MapReduce job made up of many map and reduce tasks that run against the data in HDFS spread across the DataNodes. Map tasks run on each node against the input files supplied, and reducers run to aggregate and organize the final output.

The Hadoop ecosystem has grown significantly over the years due to its extensibility. Today, the Hadoop ecosystem includes many tools and applications to help collect, store, process, analyze, and manage big data. Some of the most popular applications are:

- Spark – An open source, distributed processing system commonly used for big data workloads. Apache Spark uses in-memory caching and optimized execution for fast performance, and it supports general batch processing, streaming analytics, machine learning, graph databases, and ad hoc queries.

- Presto – An open source, distributed SQL query engine optimized for low-latency, ad-hoc analysis of data. It supports the ANSI SQL standard, including complex queries, aggregations, joins, and window functions. Presto can process data from multiple data sources including the Hadoop Distributed File System (HDFS) and Amazon S3.

- Hive – Allows users to leverage Hadoop MapReduce using a SQL interface, enabling analytics at a massive scale, in addition to distributed and fault-tolerant data warehousing.
- HBase – An open source, non-relational, versioned database that runs on top of Amazon S3 (using EMRFS) or the Hadoop Distributed File System (HDFS). HBase is a massively scalable, distributed big data store built for random, strictly consistent, real-time access for tables with billions of rows and millions of columns.
- Zeppelin – An interactive notebook that enables interactive data exploration.

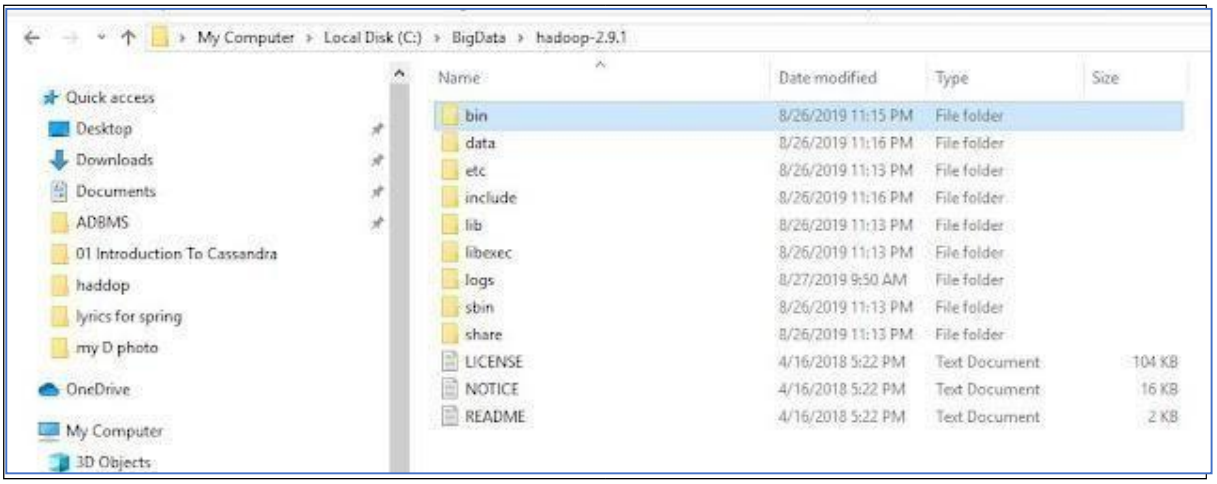
Install Hadoop 2.9.1 on Windows 10

First download the **Hadoop 2.9.1** from the below link.

https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.9.1/hadoop-2.9.1.tar.gz

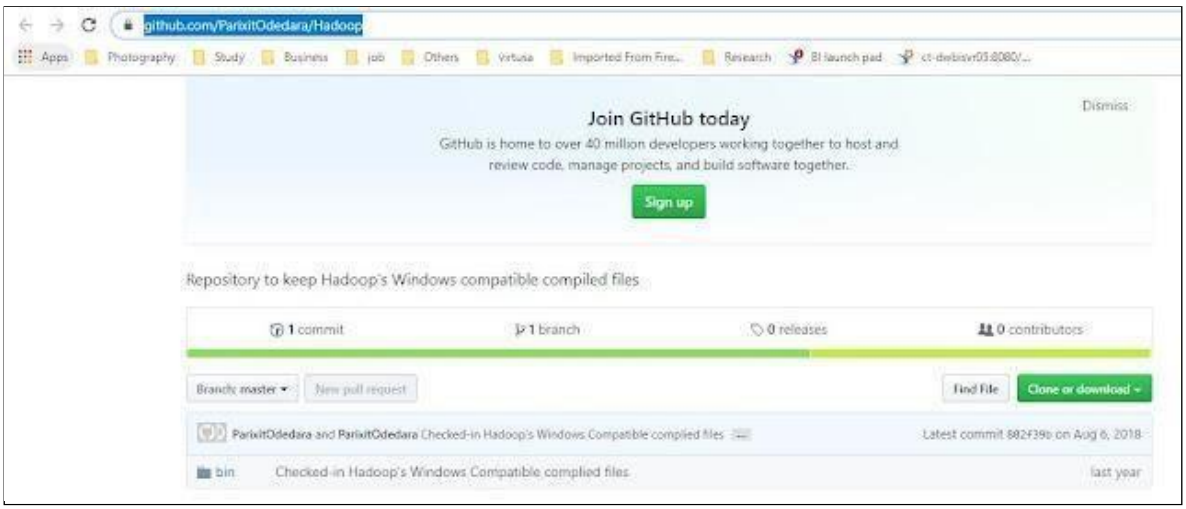
Create a folder path as below and copy the downloaded msi into this folder. Path:-

'**C:/BigData/hadoop-2.9.1**'



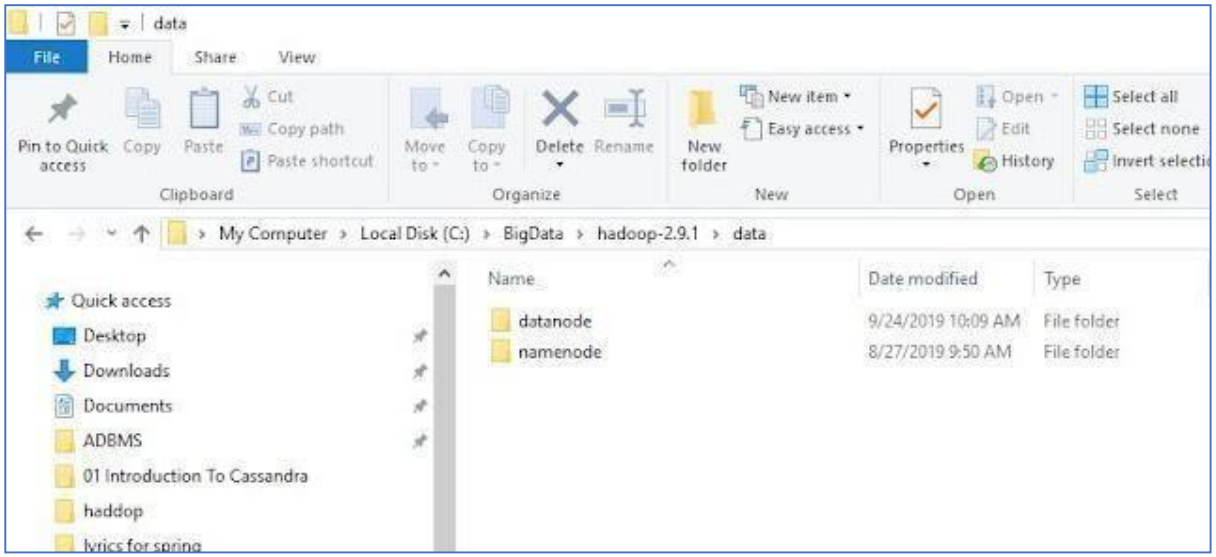
Then download the windows compatible binaries from the git hub repo.

Link:- https://github.com/ParixitOdedara/Hadoop



Extract the zip and copy all the files present under bin folder to C:\BigData\hadoop-2.9.1\bin. Replace the existing files as well.

Go to **C:/BigData/3adoop-2.9.1** and create a folder '**data**'. Inside the 'data' folder create two folders '**datanode**' and '**namenode**'.



Then Set Hadoop Environment Variables

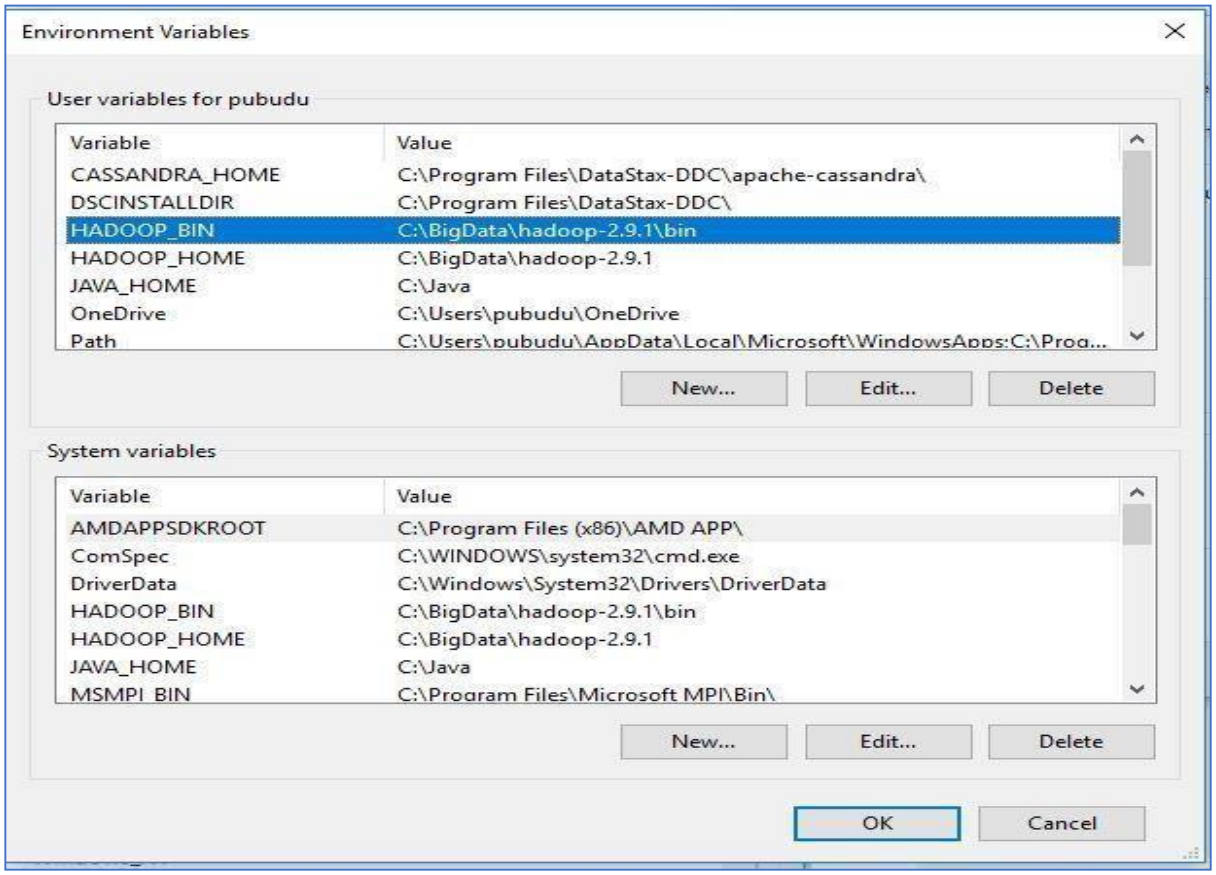
HADOOP_HOME="C:\BigData\hadoop-2.9.1"

HADOOP_BIN="C:\BigData\hadoop-2.9.1\bin"

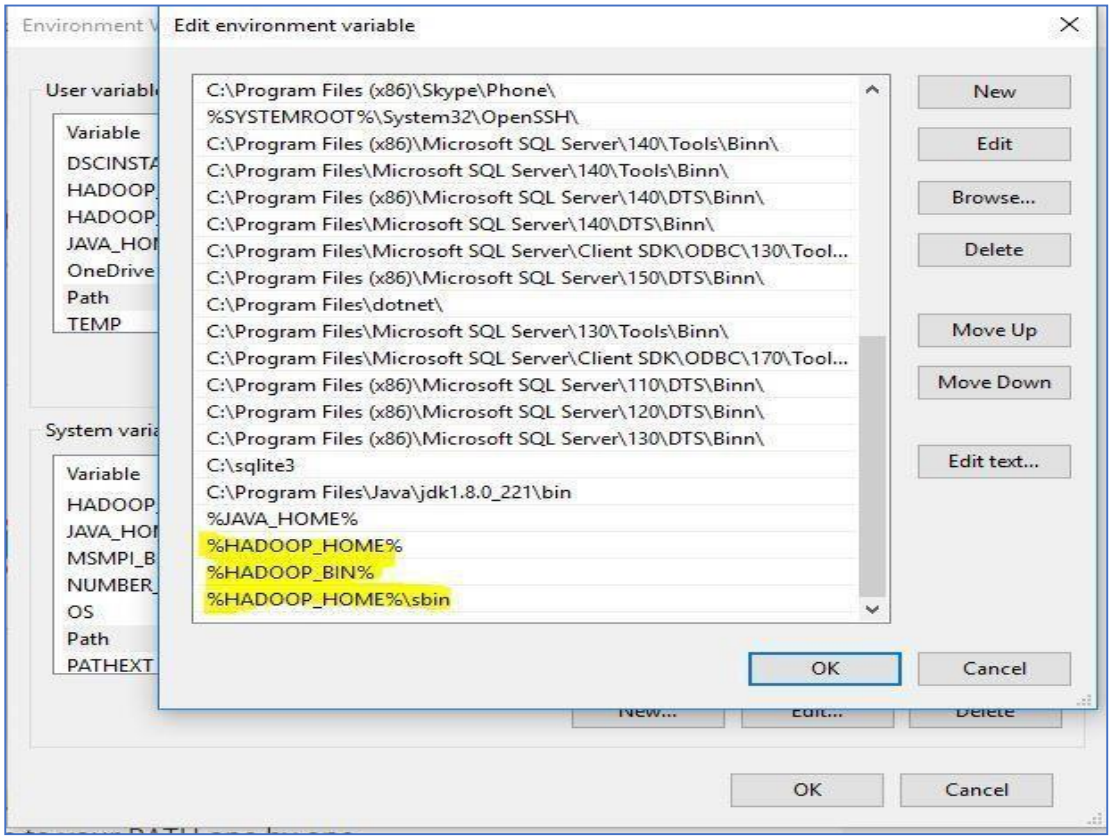
JAVA_HOME=<JDK installation location>"

To set these variables, go to My Computer or This PC. Right click --> Properties --> Advanced

System settings --> Environment variables. Click New to create a new environment variables.



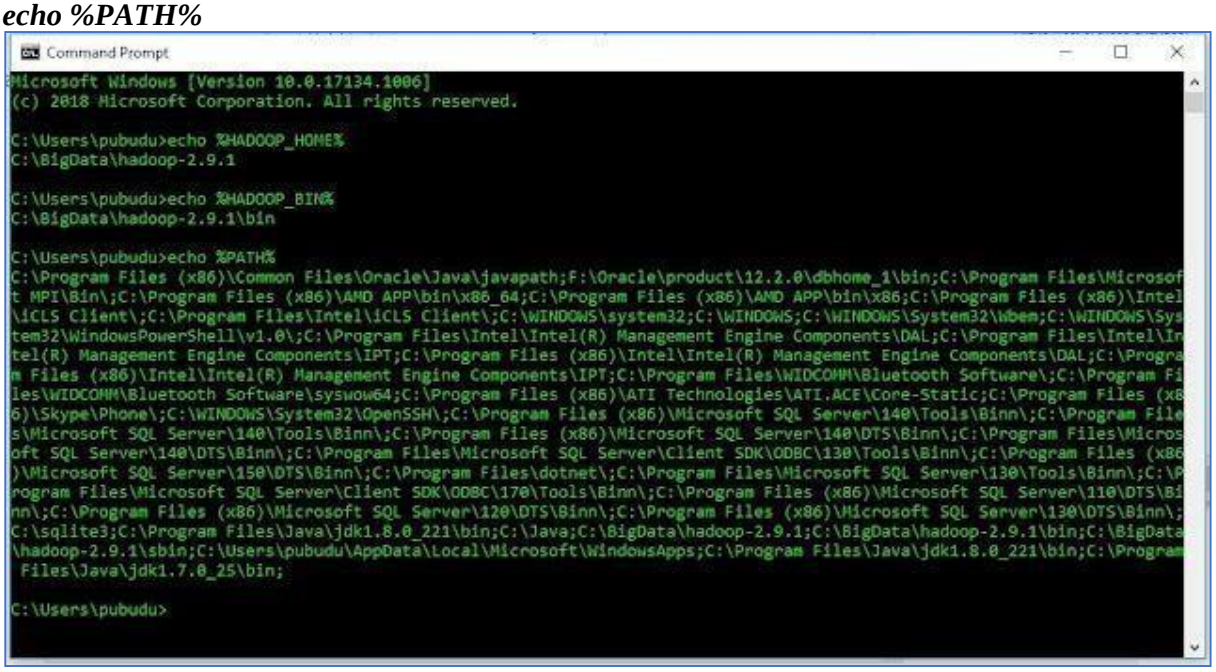
Then edit PATH Environment Variable



To validate the above setting, **open new cmd** and check the output.

echo %HADOOP_HOME%
echo %HADOOP_BIN%

echo %PATH%



To configure the Hadoop on windows we have to edit below mention files in the extracted location.

1. hadoop-env.cmd
2. core-site.xml
3. hdfs-site.xml
4. mapred-site.xml
5. yarn-site.xml

Edit hadoop-env.cmd

File location:-C:\BigData\hadoop-2.9.1\etc\hadoop\hadoop-env.cmd Need to add:-

```
set HADOOP_PREFIX=%HADOOP_HOME%
set HADOOP_CONF_DIR=%HADOOP_PREFIX%\etc\hadoop
set YARN_CONF_DIR=%HADOOP_CONF_DIR% set
PATH=%PATH%;%HADOOP_PREFIX%\bin
```

```
@rem       potential for a symlink attack.
set HADOOP_PID_DIR=%HADOOP_PID_DIR%
set HADOOP_SECURE_DN_PID_DIR=%HADOOP_PID_DIR%

@rem A string representing this instance of hadoop. %USERNAME% by default.
set HADOOP_IDENT_STRING=%USERNAME%
set HADOOP_PREFIX=%HADOOP_HOME%
set HADOOP_CONF_DIR=%HADOOP_PREFIX%\etc\hadoop
set YARN_CONF_DIR=%HADOOP_CONF_DIR%
set PATH=%PATH%;%HADOOP_PREFIX%\bin
```

Edit core-site.xml

File Location:- C:\BigData\hadoop-2.9.1\etc\hadoop\core-site.xml

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://0.0.0.0:19000</value>
  </property>
</configuration>
```

Need to add:-content within <configuration> </configuration> tags.

```
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
      <name>fs.default.name</name>
      <value>hdfs://0.0.0.0:19000</value>
    </property>
</configuration>
```

Edit hdfs-site.xml

File Location:- C:\BigData\hadoop-2.9.1\etc\hadoop\hdfs-site.xml. Need to add;- below content within <configuration> </configuration> tags.

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>C:\BigData\hadoop-2.9.1\data\namenode</value>
  </property>
  <property>
```

```
    <name>dfs.datanode.data.dir</name>
    <value>C:\BigData\hadoop-2.9.1\data\datanode</value> </property>
</configuration>
```

Edit mapred-site.xml

File location:- Open C:\BigData\hadoop-2.9.1\etc\hadoop\mapred-site.xml

Need to add:- below content within <configuration> </configuration> tags. If you don't see mapred-site.xml then open mapred-site.xml.template file and rename it to mapred-site.xml

```
 <configuration>
  <property>
    <name>mapreduce.job.user.name</name>
    <value>%USERNAME%</value>
  </property>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>yarn.apps.stagingDir</name>
    <value>/user/%USERNAME%/staging</value>
  </property>
  <property>
    <name>mapreduce.jobtracker.address</name>
    <value>local</value>
  </property>
</configuration>
```

Editing yarn-site.xml

Right click on the file, select edit and paste the following content within <configuration>

</configuration> tags.

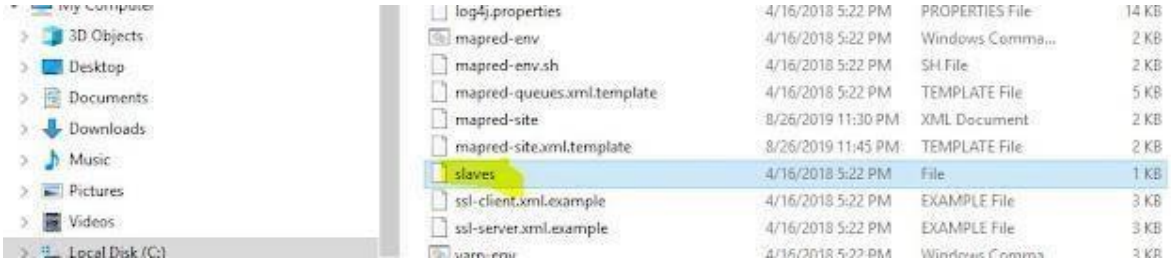
Note:- Below part already has the configuration tag, we need to copy only the part inside it.

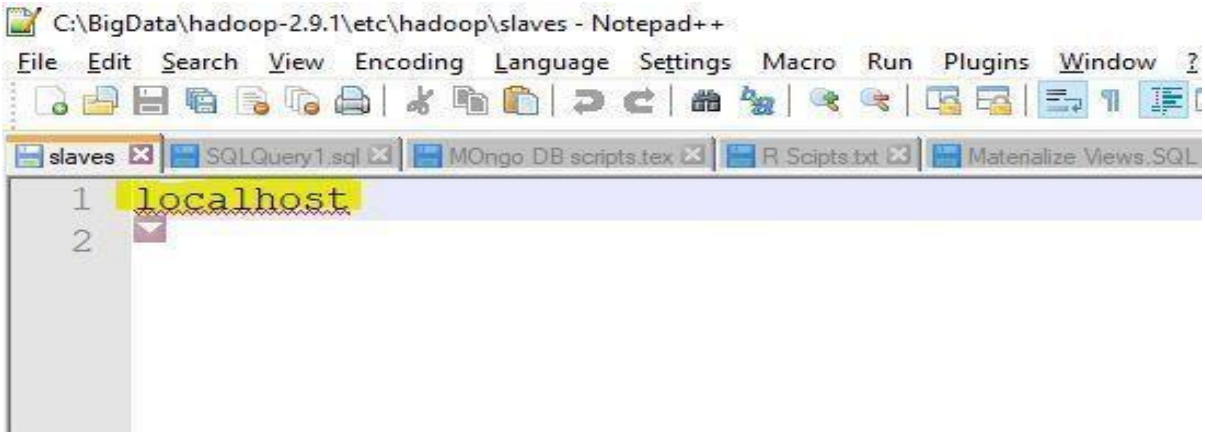
```
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
 <property>
```

```
<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
 <value>org.apache.hadoop.mapred.ShuffleHandler</value> </property>
<!-- Site specific YARN configuration properties --></configuration>
```

Additional Configuration:-

Check if C:\BigData\hadoop-2.9.1\etc\hadoop\slaves file is present, if that file not available create the file called slave and insert locahost as below.

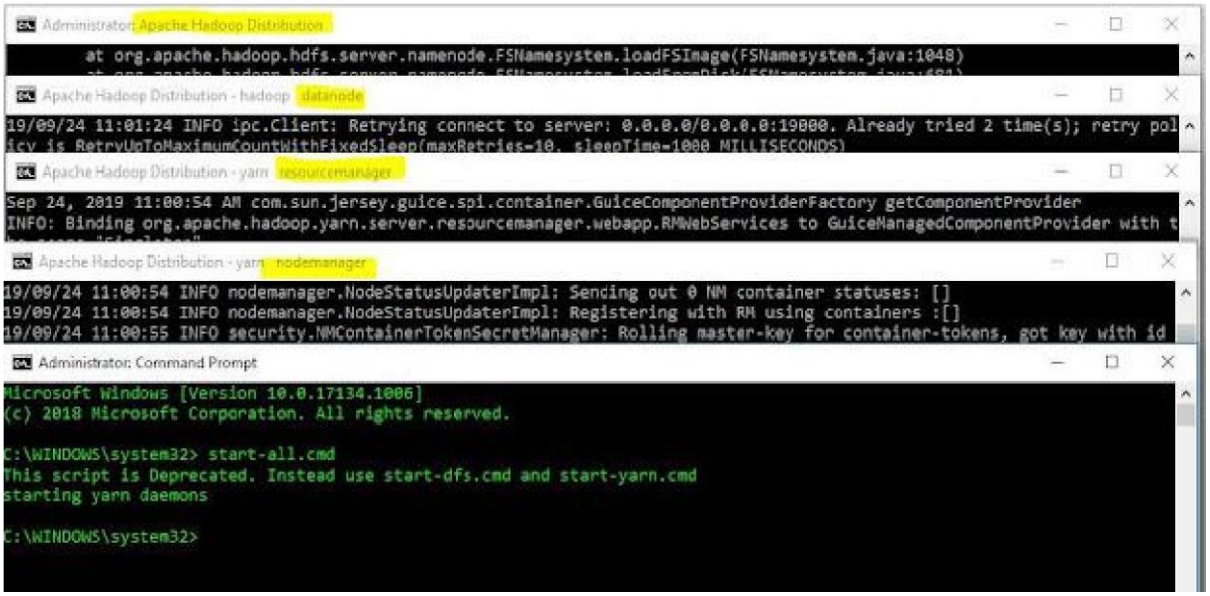




Node formatting

To format the node, **open the cmd** and execute the below command. ***hadoop namenode -format***

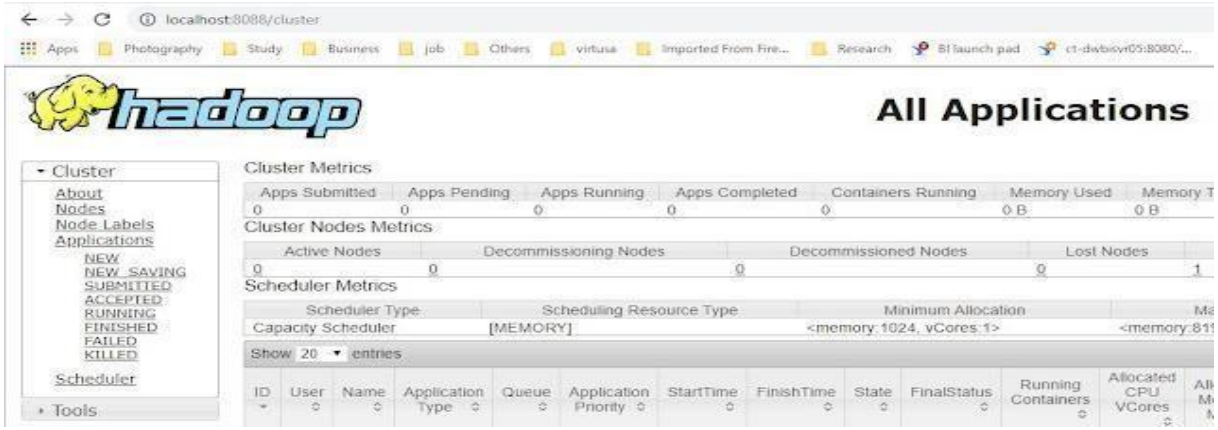
To enable the Hadoop open the **CMD as Administrator** and type below command **start-all.cmd** It will open 4 new windows cmd terminals for 4 daemon processes, namely namenode, datanode, nodemanager, and resourcemanager.



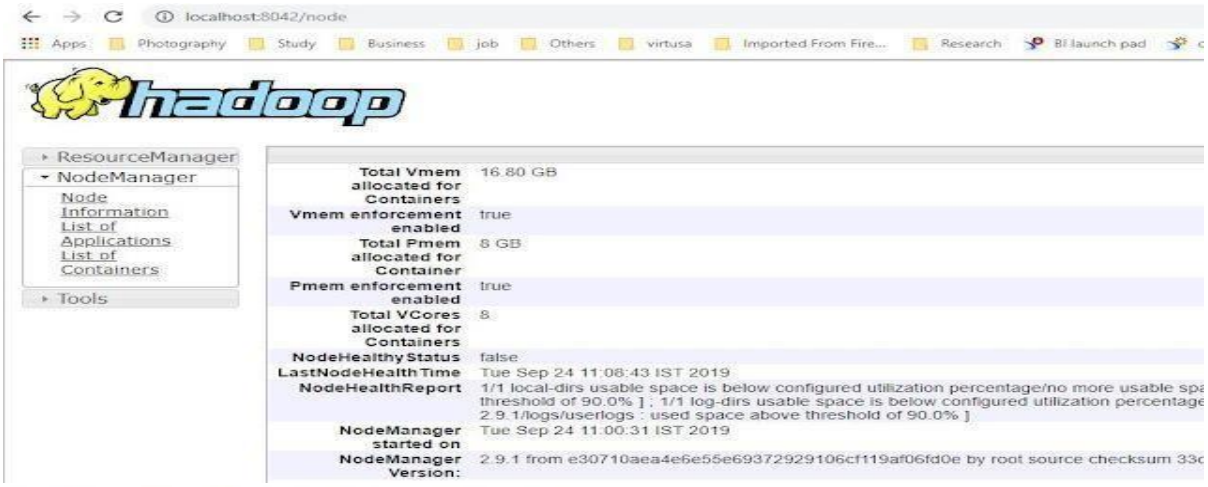
Then you have succesfully install the hadoop 2.9.1 on windows platform.

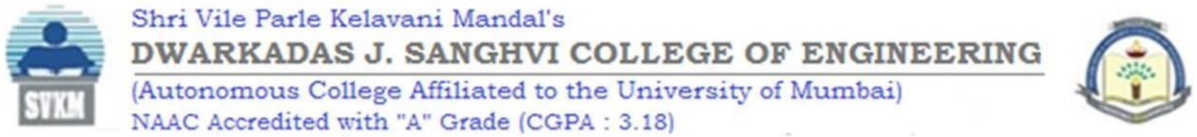
Now you can access all the Hadoop components via web urls.

To access Resource Manager go to http://localhost:8088 from your web browser.

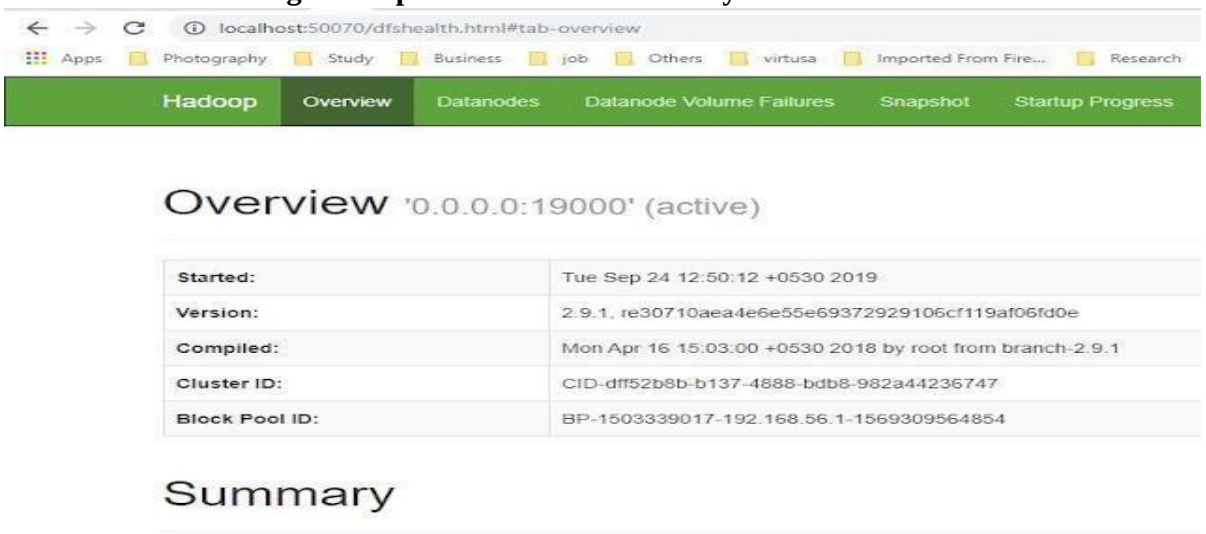


To access Node Manager go to http://localhost:8042 from your web browser.

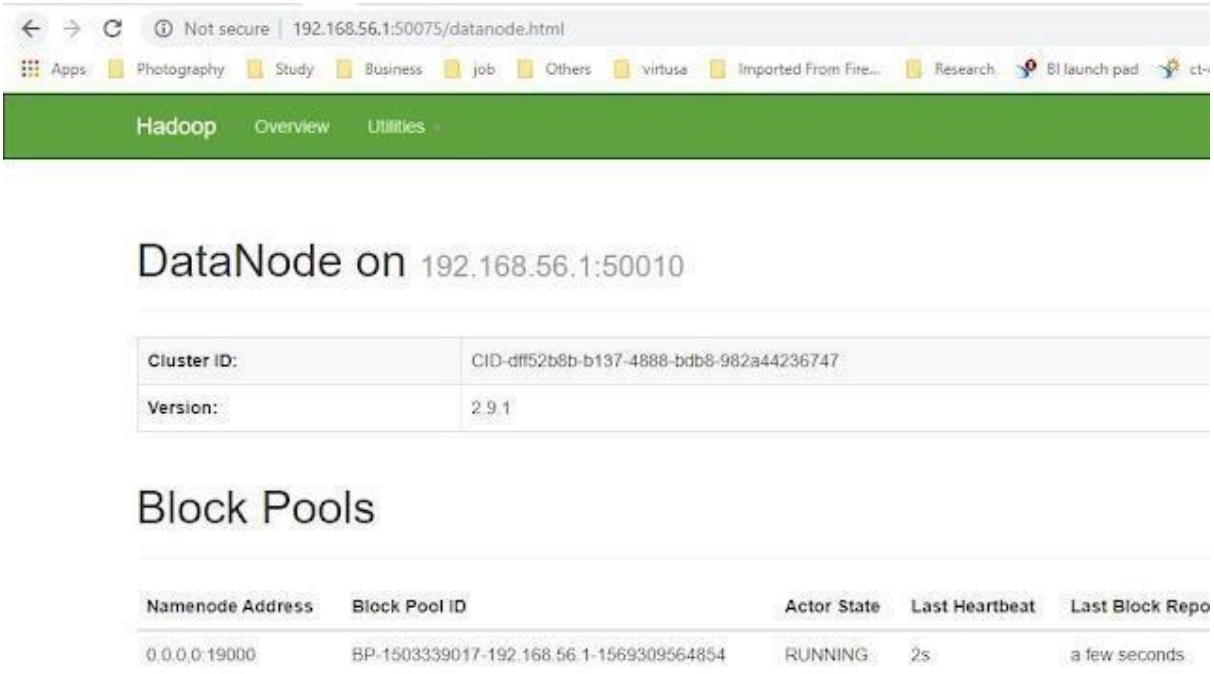




To access Name Node go to http://localhost:50070 from your web browser.



To access Data Node go to http://localhost:50075 from your web browser.



CONCLUSION: We have successfully installed Hadoop 2.9.1 on Windows 10.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 3
HDFS Commands

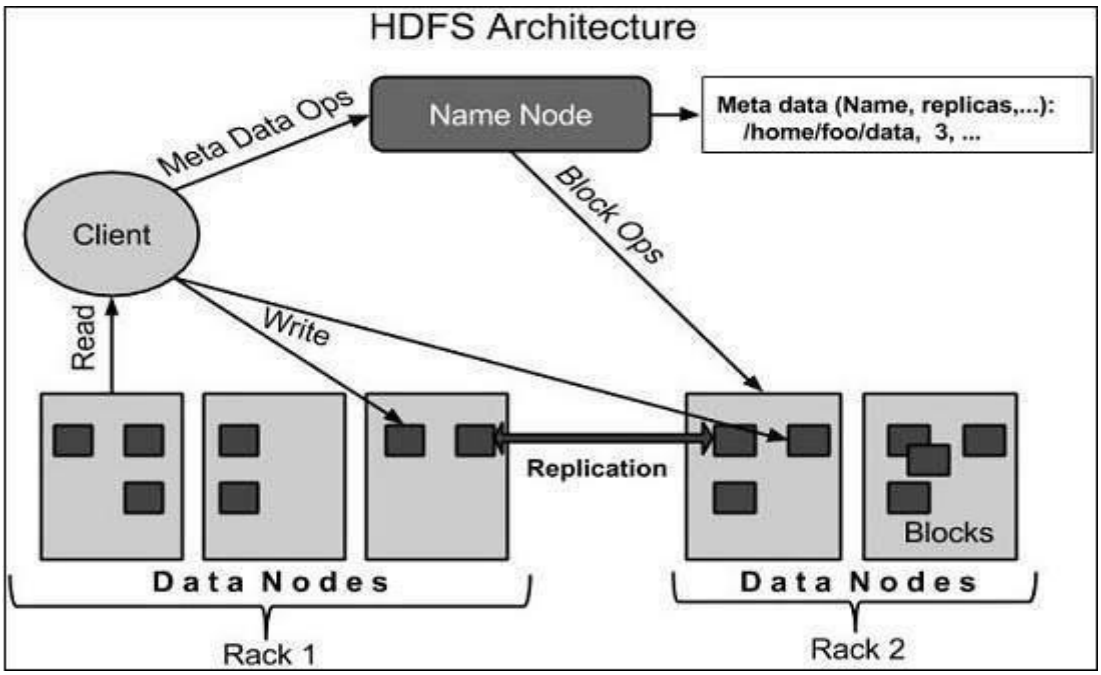
AIM: Execute different HDFS Commands.

THEORY:

Hadoop is a software framework that enables distributed storage and processing of large data sets. It consists of several open source projects, including HDFS, MapReduce, and Yarn. While Hadoop can be used for different purposes, the two most common are Big Data analytics and NoSQL database management. HDFS stands for "Hadoop Distributed File System" and is a decentralized file system that stores data across multiple computers in a cluster. This makes it ideal for large-scale storage as it distributes the load across multiple machines so there's less pressure on each individual machine. MapReduce is a programming model that allows users to write code once and execute it across many servers. When combined with HDFS, MapReduce can be used to process massive data sets in parallel by dividing work up into smaller chunks and executing them simultaneously.

HDFS Architecture

HDFS is an Open source component of the Apache Software Foundation that manages data. HDFS has scalability, availability, and replication as key features. Name nodes, secondary name nodes, data nodes, checkpoint nodes, backup nodes, and blocks all make up the architecture of HDFS. HDFS is fault-tolerant and is replicated. Files are distributed across the cluster systems using the Name node and Data Nodes. The primary difference between Hadoop and Apache HBase is that Apache HBase is a non-relational database and Apache Hadoop is a non-relational data store.



HDFS follows the master-slave architecture and it has the following elements.

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks −

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

HDFS Commands

After successful installation of Hadoop, we execute different HDFS Commands.

Open a new Windows Command Prompt and run below commands. C:\hadoop-2.9.1\hadoop-2.9.1>cd bin

ls: This command is used to list all the files. Use lsr for recursive approach. It is useful when we want a hierarchy of a folder.

C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -ls /

```
Found 2 items
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 20:06 /sampledir
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 19:57 /test
```

mkdir: To create a directory. In Hadoop *dfs* there is no home directory by default.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -mkdir /user C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -mkdir /user/Lenovo
```

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -ls / Found
3 items
drwxr-xr-x - LENOVO supergroup    0 2023-02-27 20:06 /sampledir
drwxr-xr-x - LENOVO supergroup    0 2023-02-27 19:57 /test
```

```
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 20:23 /user
```

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -lsr /user
lsr: DEPRECATED: Please use 'ls -R' instead.
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 20:23 /user/Lenovo
```

touchz: It creates an empty file.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -touchz /user/myfile.txt C:\hadoop-2.9.1\hadoop-
2.9.1\bin>hdfs dfs -ls -R /user
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 20:23 /user/Lenovo
-rw-r--r-- 1 LENOVO supergroup          0 2023-02-27 20:26 /user/myfile.txt
```

copyFromLocal (or) put: To copy files/folders from local file system to hdfs store. This is the most important command. Local filesystem means the files present on the OS.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -put C:/hadoop-2.9.1/hadoop-2.9.1/Sample.txt /user
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -ls -R /user drwxr-xr-x - LENOVO supergroup          0
2023-02-27 20:23 /user/Lenovo -rw-r--r-- 1 LENOVO supergroup  12 2023-02-27 20:30
/user/Sample.txt -rw-r--r-- 1 LENOVO supergroup          0 2023-02-27 20:26 /user/myfile.txt
```

cat: To print file contents.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -cat /user/Sample.txt Hello Hadoop
```

copyToLocal (or) get: To copy files/folders from hdfs store to local file system.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -get /user/Sample.txt ../HadoopExamples
```

cp: This command is used to copy files within hdfs.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -mkdir /user_copied C:\hadoop-2.9.1\hadoop-
2.9.1\bin>hdfs dfs -cp /user /user_copied C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -ls
/user_copied
```

```
Found 1 items
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 20:48 /user_copied/user
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -ls /user Found 3 items
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 20:23 /user/Lenovo
```

```
-rw-r--r-- 1 LENOVO supergroup          12 2023-02-27 20:30 /user/Sample.txt
-rw-r--r-- 1 LENOVO supergroup          0 2023-02-27 20:26 /user/myfile.txt
```

mv: This command is used to move files within hdfs.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -mv /user/myfile.txt /user_copied
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -ls /user Found
2 items
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 20:23 /user/Lenovo
-rw-r--r-- 1 LENOVO supergroup          12 2023-02-27 20:30 /user/Sample.txt
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -ls /user_copied
Found 2 items
-rw-r--r-- 1 LENOVO supergroup       0 2023-02-27 20:26 /user_copied/myfile.txt
drwxr-xr-x - LENOVO supergroup       0 2023-02-27 20:48 /user_copied/user
```

rmr: This command deletes a file from HDFS *recursively*. It is very useful command when you want to delete a *non-empty directory*.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -rmr /user_copied
rmr: DEPRECATED: Please use '-rm -r' instead.
```

```
Deleted /user_copied C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -
ls /user_copied ls: `/user_copied': No such file or directory
```

du: It will give the size of each file in directory.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -du /user 0 /user/Lenovo 12 /user/Sample.txt
```

dus: This command will give the total size of directory/file.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -dus /user dus: DEPRECATED:
Please use 'du -s' instead.
12  /user
```

stat: It will give the last modified time of directory or path. In short it will give stats of the directory or file.

```
C:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -stat /user 2023-02-27 15:20:27
```

setrep: This command is used to change the replication factor of a file/directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs *core-site.xml*).

```
c:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -setrep -R 4 /user
Replication 4 set: /user/Sample.txt
c:\hadoop-2.9.1\hadoop-2.9.1\bin>hdfs dfs -ls /user Found
2 items
drwxr-xr-x - LENOVO supergroup          0 2023-02-27 20:23 /user/Lenovo
-rw-r--r-- 4 LENOVO supergroup         12 2023-02-27 20:30 /user/Sample.txt
```

CONCLUSION: We have successfully execute different HDFS Commands.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 4
HIVE COMMANDS

AIM: Execute HIVE commands to load, insert, retrieve, update, or delete data in the tables.

THEORY:
Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.
Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not

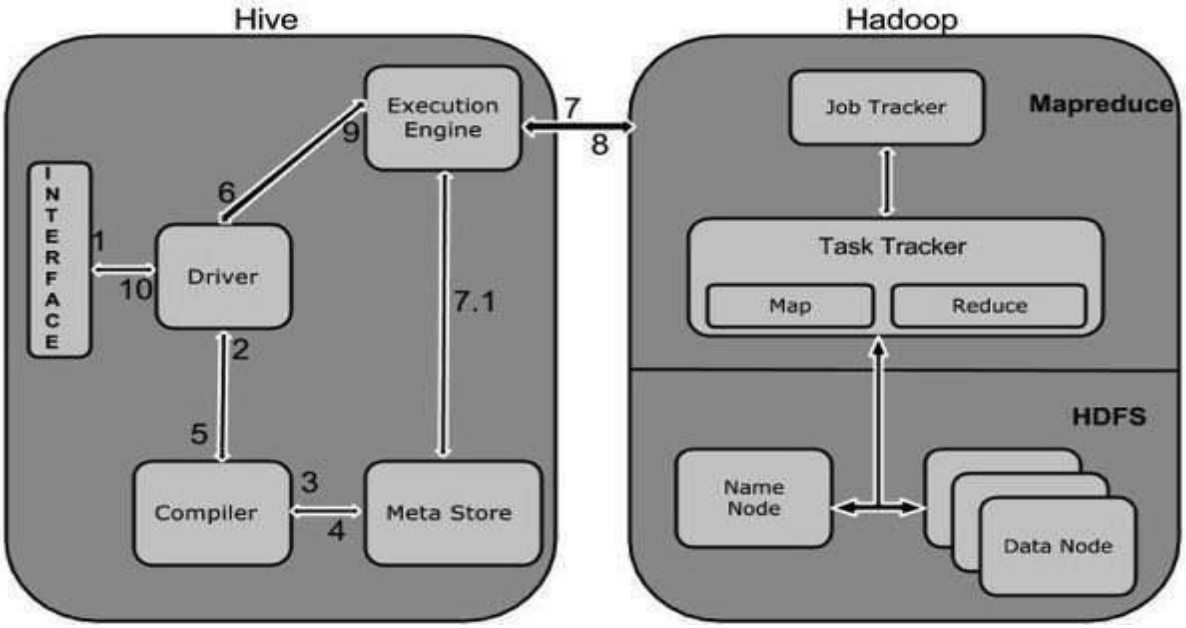
- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

Features of Hive

- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL. • It is familiar, fast, scalable, and extensible.

Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.



The following table defines how Hive interacts with Hadoop framework:

1	**Execute Query** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.

Step No.	**Operation**

2	**Get Plan** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	**Get Metadata** The compiler sends metadata request to Metastore (any database).
4	**Send Metadata** Metastore sends metadata as a response to the compiler.
5	**Send Plan** The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6	**Execute Plan** The driver sends the execute plan to the execution engine.
7	**Execute Job** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
7.1	**Metadata Ops** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	**Fetch Result** The execution engine receives the results from Data nodes.
9	**Send Results** The execution engine sends those resultant values to the driver.
10	**Send Results** The driver sends the results to Hive Interfaces.

CONCLUSION: We have successfully executed HIVE Queries using HQL in Cloudera Framework.

HIVE Query Language

http://127.0.0.1:4200 sandbox login: root
root@sandbox.hortonworks.com's
password:
Last login: Thu Mar 9 06:30:54 2023 from 172.17.0.2
[root@sandbox ~]# hive

Logging initialized using configuration in file:/etc/hive/2.5.0.0-1245/0/hive-log4j.properties

Show databases already existing in Hive

```
hive> show
databases; OK
bdiexample default
foodmart
retail
xademo
Time taken: 0.025 seconds, Fetched: 5 row(s)
```

Creating a new database

```
hive> create database studentdb;
```

```
OK
Time taken: 0.072 seconds
```

Using the database for executing queries

```
hive> use studentdb;
OK
Time taken: 0.042 seconds
```

Creating a table in Hive

The most used optional clauses in creating a table are:

- IF NOT EXISTS – You can use IF NOT EXISTS to avoid the error in case the table is already present. Hive checks if the requesting table already presents,
- EXTERNAL – Used to create external table
- TEMPORARY – Used to create temporary table.
- ROW FORMAT – Specifies the format of the row.
- FIELDS TERMINATED BY – By default Hive use **^A** field separator, To load a file that has a custom field separator like comma, pipe, tab use this option.
- PARTITION BY – Used to create partition data. Using this improves performance.
- CLUSTERED BY – Dividing the data into a specific number for buckets.
- LOCATION – You can specify the custom location where to store the data on HDFS.
- The *Optimized Row Columnar* (ORC) file format provides a highly efficient way to store Hive data. Using ORC files improves performance when Hive is reading, writing, and processing data. Other file formats supported are : JSON, Text, Sequence, etc.

```
hive> create table student(id int, name varchar(20), branch varchar(20), mobile int)
> PARTITIONED BY (load_date date)
> CLUSTERED BY(id) INTO 3 BUCKETS
> STORED AS ORC TBLPROPERTIES ('transactional'='true');
OK
Time taken: 0.657 seconds
```

#Inserting values in the table

```
hive> insert into student partition (load_date = '2023-03-09') values
(101,'Manav','Computer',123456);
```

```
Query ID = root_20230309064202_db1b9f50-234f-42ed-b028-22d9ba0ff933
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1678329811540_0003)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
```

VERTICES: 01/01 [==========>>] 100% ELAPSED TIME: 5.54 s

Loading data to table studentdb.student partition (load_date=2023-03-09)
Partition studentdb.student{load_date=2023-03-09} stats: [numFiles=1, numRows=0, totalSize=858, rawDataSize=0]
OK
Time taken: 10.075 seconds

```
hive> insert into student partition (load_date = '2023-03-09') values (102,'Devraj','Computer',341256
);
```

Query ID = root_20230309064238_73e473b3-adc1-4fe9-818c-e077dbed113c
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1678329811540_0003)

VERTICES	STATUS	TOTAL	COMPLETED	RUNNING	PENDING	FAILED	KILLED
Map 1	SUCCEEDED	1	1	0	0	0	0

VERTICES: 01/01 [==========>>] 100% ELAPSED TIME: 4.71 s

Loading data to table studentdb.student partition (load_date=2023-03-09)
Partition studentdb.student{load_date=2023-03-09} stats: [numFiles=2, numRows=0, totalSize=1720, rawD ataSize=0]
OK
Time taken: 5.983 seconds

To display contents of a table

```
hive> select * from student;
OK
101    Manav Computer        123456 2023-03-09
102    Devraj Computer       341256 2023-03-09
```

Time taken: 0.216 seconds, Fetched: 2 row(s)

Order By Clause (Ordering the result in descending order; by default ascending order)

```
hive> select id, name from student order by id desc;
```

Query ID = root_20230309064323_da0caba1-19a2-433e-b579-4505f17dcabc
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1678329811540_0003)

```
--------------------------------------------------------------------------------
      VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......SUCCEEDED       1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 16.65 s
--------------------------------------------------------------------------------

OK
102     Devraj
101     Manav
Time taken: 17.594 seconds, Fetched: 2 row(s)
```

Inserting a new tuple in the table

```
hive> insert into student partition (load_date = '2023-03-09') values (103,'Sahil','Mechanical',98762
3);
Query ID = root_20230309064439_bfdf6f3d-ce54-4cf6-b31e-65bc23b689e1
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1678329811540_0003)

--------------------------------------------------------------------------------
      VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 3.67 s
--------------------------------------------------------------------------------

Loading data to table studentdb.student partition (load_date=2023-03-09)
Partition studentdb.student{load_date=2023-03-09} stats: [numFiles=3, numRows=0, totalSize=2579,
rawD
ataSize=0]
OK
Time taken: 5.262 seconds
```

Display contents of the table after new entry

```
hive> select * from student;
OK
101     Manav Computer          123456 2023-03-09
102     Devraj Computer         341256 2023-03-09
103     Sahil Mechanical        987623 2023-03-09
Time taken: 0.121 seconds, Fetched: 3 row(s)
```

Group By Clause (Grouping the result w.r.t branch here)

```
hive> SELECT branch,count(*) FROM student GROUP BY branch;
```

```
Query ID = root_20230309064938_1c36c2cc-682f-4956-bed3-4897f9bd3be4 Total
jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1678329811540_0003)

----------------------------------------------------------------------
      VERTICES       STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
----------------------------------------------------------------------

 Map 1 ..... SUCCEEDED      3        3       0       0       0       0
Reducer 2 .....SUCCEEDED      1        1       0       0       0       0
----------------------------------------------------------------------

VERTICES: 02/02 [==========================>>] 100% ELAPSED TIME: 20.15 s
----------------------------------------------------------------------


OK

Computer        2
Mechanical      1
Time taken: 21.075 seconds, Fetched: 2 row(s)
```

Update the value in table

Note: update, delete statements will work only if during the creation of table transactional property is set to 'true'.

```
hive> update student set name='Sayli' where id = 103;
Query ID = root_20230309070310_dee26faf-f65d-4c8a-b966-bc18cc118ccc
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1678329811540_0004)

----------------------------------------------------------------------
      VERTICES       STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
----------------------------------------------------------------------

 Map 1 ..... SUCCEEDED      3        3       0       0       0       0
Reducer 2 .....SUCCEEDED      1        1       0       0       0       0
----------------------------------------------------------------------

VERTICES: 02/02 [==========================>>] 100% ELAPSED TIME: 26.39 s
----------------------------------------------------------------------


Loading data to table studentdb.student partition (load_date=null)
        Time taken for load dynamic partitions : 464
       Loading partition {load_date=2023-03-09}
        Time taken for adding to write entity : 5
Partition studentdb.student{load_date=2023-03-09} stats: [numFiles=4, numRows=0, totalSize=3457,
rawDataSize=0]
OK
```

Time taken: 36.217 seconds

```
hive> select * from student;

OK
101   Manav Computer          123456 2023-03-09
102   Devraj Computer         341256 2023-03-09
103   Sayli Mechanical        987623 2023-03-09
      Time taken: 0.262 seconds, Fetched: 3 row(s)
```

#Delete the entry from the table

```
hive> delete from student where name = 'Sayli';
Query ID = root_20230309070624_382e39b5-755b-41c2-8f04-362939800cbf
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1678329811540_0004)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... SUCCEEDED      3          3        0        0       0       0
Reducer 2 ..... SUCCEEDED       1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 19.68 s
----------------------------------------------------------------------------------------------

Loading data to table studentdb.student partition (load_date=null)
        Time taken for load dynamic partitions : 516
        Loading partition {load_date=2023-03-09}
        Time taken for adding to write entity : 0
Partition studentdb.student{load_date=2023-03-09} stats: [numFiles=5, numRows=0, totalSize=3990,
rawDataSize=0]
OK
Time taken: 22.826 seconds
```

```
hive> select * from student;
OK
101     Manav   Computer        123456 2023-03-09
102     Devraj Computer         341256 2023-03-09 Time taken: 0.319 seconds, Fetched: 2 row(s)
```

Rename a table using the ALTER statement

```
hive> alter table student rename to stud;
OK
Time taken: 0.539 seconds
```

Check whether table is renamed

```
hive> show tables;
OK
stud
values__tmp___table__1
values__tmp___table__2
values_tmp_table_3
Time taken: 0.119 seconds, Fetched: 4 row(s)
```

Add new column in the table using ALTER statement

```
hive> alter table stud add columns(cgpa double);
OK
Time taken: 0.441 seconds
```

CGPA column is added. Initially its value in all rows will be NULL as we haven't entered CGPA value.

```
hive> select * from stud;
OK
101     Manav Computer          123456 NULL 2023-03-09
102     Devraj   Computer       341256 NULL 2023-03-09
Time taken: 0.241 seconds, Fetched: 2 row(s)
```

Changing Column Name with ALTER statement and also changing the datatype from varchar to String.

```
hive> ALTER TABLE stud CHANGE name sname String;
OK
Time taken: 0.478 seconds
```

Updating the CGPA values in table (i.e. changing the NULL Value)

```
hive> update stud set cgpa = 7.9 where id = 101;
Query ID = root_20230309071806_94217582-6ff2-408e-addf-3af8a3655e94
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening... Session
re-established.
Status: Running (Executing on YARN cluster with App id application_1678329811540_0005)

----------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------
Map 1 .......... SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 33.39 s
----------------------------------------------------------------------
```

```
Loading data to table studentdb.stud partition (load_date=null)
        Time taken for load dynamic partitions : 417
       Loading partition {load_date=2023-03-09}
        Time taken for adding to write entity : 0
Partition  studentdb.stud{load_date=2023-03-09}  stats:  [numFiles=6,  numRows=0,  totalSize=4956,
rawDataSize=0]
OK
Time taken: 45.136 seconds
```

```
hive> update stud set cgpa = 8.8 where id = 102;
Query ID = root_20230309071919_6f74a8dc-b5a9-406b-81af-fea4bc3a09e8
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1678329811540_0005)

----------------------------------------------------------------------------
     VERTICES       STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
----------------------------------------------------------------------------
Map 1 .......... SUCCEEDED      3        3       0       0       0       0
Reducer 2 ........SUCCEEDED      1        1       0       0       0       0
----------------------------------------------------------------------------
VERTICES: 02/02 [==========================>>] 100% ELAPSED TIME: 12.69 s
----------------------------------------------------------------------------

Loading data to table studentdb.stud partition (load_date=null)
        Time taken for load dynamic partitions : 183
       Loading partition {load_date=2023-03-09}
        Time taken for adding to write entity : 1
Partition  studentdb.stud{load_date=2023-03-09}  stats:  [numFiles=7,  numRows=0,  totalSize=5926,
rawDataSize=0]
OK
Time taken: 14.739 seconds
```

Check whether the CGPA values are updated.

```
hive> select * from stud;

OK
101     Manav Computer          123456 7.9      2023-03-09
102     Devraj Computer 341256 8.8      2023-03-09
Time taken: 0.196 seconds, Fetched: 2 row(s)
```

Performing Aggregate Functions (SUM, MAX, MIN, AVG, COUNT) on table

```
hive> select sum(cgpa) from stud;
```

```
Query ID = root_20230309074116_00e58970-8700-4fd9-897b-15e8ce2a3d5c
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.

Status: Running (Executing on YARN cluster with App id application_1678329811540_0006)
```

```
--------------------------------------------------------------------------------
      VERTICES      STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
--------------------------------------------------------------------------------
Map 1 .......... SUCCEEDED      3        3       0       0       0       0
Reducer 2 ........SUCCEEDED      1        1       0       0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02 [==========================>>] 100% ELAPSED TIME: 25.89 s
--------------------------------------------------------------------------------

OK
16.700000000000003
Time taken: 39.321 seconds, Fetched: 1 row(s)

hive> select max(cgpa) from stud;
Query ID = root_20230309074240_a75dc01e-f8c6-4694-b6d1-45012b7652e6
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1678329811540_0006)

--------------------------------------------------------------------------------
      VERTICES      STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
--------------------------------------------------------------------------------
Map 1 .......... SUCCEEDED      3        3       0       0       0       0
Reducer 2 ........SUCCEEDED      1        1       0       0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02 [==========================>>] 100% ELAPSED TIME: 12.20 s
--------------------------------------------------------------------------------

OK
8.8
Time taken: 14.61 seconds, Fetched: 1 row(s)

hive> select min(cgpa) from stud;
Query ID = root_20230309074335_8d8c7447-1684-4fd0-b3b2-06db2cb8a439
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1678329811540_0006)

--------------------------------------------------------------------------------
      VERTICES      STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
--------------------------------------------------------------------------------
Map 1 .......... SUCCEEDED      3        3       0       0       0       0
Reducer 2 ........SUCCEEDED      1        1       0       0       0       0
```

VERTICES: 02/02 [==========>>] 100% ELAPSED TIME: 18.26 s

OK

7.9

Time taken: 19.969 seconds, Fetched: 1 row(s)

```
hive> select avg(cgpa) from stud;
```

Query ID = root_20230309074547_64a6e36d-247e-418c-88d1-c1ebc565e734
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1678329811540_0006)

VERTICES	STATUS	TOTAL	COMPLETED	RUNNING	PENDING	FAILED	KILLED
Map 1	SUCCEEDED	3	3	0	0	0	0
Reducer 2	SUCCEEDED	1	1	0	0	0	0

VERTICES: 02/02 [==========>>] 100% ELAPSED TIME: 16.41 s

OK

8.350000000000001

Time taken: 18.007 seconds, Fetched: 1 row(s)

```
hive> select count(cgpa) from stud;
```

Query ID = root_20230309074649_0342e8d0-ab88-40a0-9306-16aca6cdb51e
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1678329811540_0006)

VERTICES	STATUS	TOTAL	COMPLETED	RUNNING	PENDING	FAILED	KILLED
Map 1	SUCCEEDED	3	3	0	0	0	0
Reducer 2	SUCCEEDED	1	1	0	0	0	0

VERTICES: 02/02 [==========>>] 100% ELAPSED TIME: 8.17 s

OK

2

Time taken: 9.295 seconds, Fetched: 1 row(s)

Operators in HIVE (Arithmetic, Relational, Logical)

Relational Operators

These operators are used to compare two operands. The following table describes the relational operators available in Hive:

Operator	Operand	Description
A = B	all primitive types	TRUE if expression A is equivalent to expression B otherwise FALSE.
A != B	all primitive types	TRUE if expression A is not equivalent to expression B otherwise FALSE.
A < B	all primitive types	TRUE if expression A is less than expression B otherwise FALSE.
A <= B	all primitive types	TRUE if expression A is less than or equal to expression B otherwise FALSE.
A > B	all primitive types	TRUE if expression A is greater than expression B otherwise FALSE.
A >= B	all primitive types	TRUE if expression A is greater than or equal to expression B otherwise FALSE.
A IS NULL	all types	TRUE if expression A evaluates to NULL otherwise FALSE.
A IS NOT NULL	all types	FALSE if expression A evaluates to NULL otherwise TRUE.
A LIKE B	Strings	TRUE if string pattern A matches to B otherwise FALSE.
A RLIKE B	Strings	NULL if A or B is NULL, TRUE if any substring of A matches the Java regular expression B , otherwise FALSE.
A REGEXP B	Strings	Same as RLIKE.

Arithmetic Operators

These operators support various common arithmetic operations on the operands. All of them return number types. The following table describes the arithmetic operators available in Hive:

Operators	Operand	Description
A + B	all number types	Gives the result of adding A and B.
A - B	all number types	Gives the result of subtracting B from A.
A * B	all number types	Gives the result of multiplying A and B.
A / B	all number types	Gives the result of dividing B from A.
A % B	all number types	Gives the reminder resulting from dividing A by B.
A & B	all number types	Gives the result of bitwise AND of A and B.
A | B	all number types	Gives the result of bitwise OR of A and B.
A ^ B	all number types	Gives the result of bitwise XOR of A and B.
~A	all number types	Gives the result of bitwise NOT of A.

Logical Operators

The operators are logical expressions. All of them return either TRUE or FALSE.

Operators	Operands	Description
A AND B	boolean	TRUE if both A and B are TRUE, otherwise FALSE.

A && B	boolean	Same as A AND B.
A OR B	boolean	TRUE if either A or B or both are TRUE, otherwise FALSE.
A || B	boolean	Same as A OR B.
NOT A	boolean	TRUE if A is FALSE, otherwise FALSE.
!A	boolean	Same as NOT A.

```
hive> select * from stud where cgpa < 9.0;
OK
101     Manav Computer          123456 7.9      2023-03-09
102     Devraj Computer         341256 8.8      2023-03-09
        Time taken: 0.473 seconds, Fetched: 2 row(s)
```

```
hive> select * from stud where cgpa >= 7.5;
OK
101     Manav Computer          123456 7.9      2023-03-09
102     Devraj Computer         341256 8.8      2023-03-09
        Time taken: 0.219 seconds, Fetched: 2 row(s)
```

Joins in HIVE

Basically, for combining specific fields from two tables by using values common to each one we use Hive JOIN clause.

In other words, to combine records from two or more tables in the database we use JOIN clause. **Types of Joins in Hive**

1. **Inner join in Hive 2. Left Outer Join in Hive 3. Right Outer Join in Hive 4. Full Outer Join in Hive**

a. Inner Join

Basically, to combine and retrieve the records from multiple tables we use Hive Join clause. Moreover, by **using the primary keys and foreign keys** of the tables JOIN condition is to be raised.

b. Left Outer Join

On defining HiveQL Left Outer Join, even if there are no matches in the right table it returns all the rows from the left table.

To be more specific, even if the ON clause matches 0 (zero) records in the right table, then also this Hive JOIN still returns a row in the result. Although, it returns with NULL in each column from the right table.

In addition, it returns all the values from the left table. Also, the matched values from the right table, or NULL in case of no matching JOIN predicate.

c. Right Outer Join

Basically, even if there are no matches in the left table, HiveQL Right Outer Join returns all the rows from the right table.

To be more specific, even if the ON clause matches 0 (zero) records in the left table, then also this Hive JOIN still returns a row in the result. Although, it returns with NULL in each column from the left table.

In addition, it returns all the values from the right table. Also, the matched values from the left table or NULL in case of no matching join predicate.

d. Full Outer Join

The major purpose of this HiveQL Full outer Join is it combines the records of both the left and the right outer tables which fulfills the Hive JOIN condition. Moreover, this joined table contains either all the records from both the tables or fills in NULL values for missing matches on either side.

Creating and inserting values in 2 tables namely, customers and orders to execute JOINS in HIVE.

```
hive> create table customers(id int, name String, age int, address String, salary int)
> partitioned by (load_date date)
> clustered by(id) into 3 buckets
> stored as orc tblproperties ('transactional'='true');
OK
Time taken: 0.842 seconds
```

```
hive> insert into customers partition (load_date = '2023-03-09') values
(1,"Ross",25,"Mumbai",25000);
```

```
Query ID = root_20230309080253_358e5b90-f99b-4fc3-a2cd-41571ab6b21c
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1678329811540_0007)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
```

VERTICES: 01/01 [==========================>>] 100% ELAPSED TIME: 5.21 s

```
----------------------------------------------------------------------------
Loading data to table studentdb.customers partition (load_date=2023-03-09)
Partition studentdb.customers{load_date=2023-03-09} stats: [numFiles=1, numRows=0, totalSize=865, rawDataSize=0]
OK
Time taken: 14.342 seconds
```

```
hive> insert into customers partition (load_date = '2023-03-09') values
(2,"Mike",27,"Bhopal",35000);
```

```
Query ID = root_20230309080416_207775fb-ecc4-40c0-a216-281a829b9581
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1678329811540_0007)
```

```
--------------------------------------------------------------------------------
      VERTICES       STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
--------------------------------------------------------------------------------
 Map 1 .......... SUCCEEDED      1        1       0       0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01 [==========================>>] 100% ELAPSED TIME: 7.01 s
--------------------------------------------------------------------------------

Loading data to table studentdb.customers partition (load_date=2023-03-09)
Partition studentdb.customers{load_date=2023-03-09} stats: [numFiles=2, numRows=0, totalSize=1735, rawDataSize=0]
OK
Time taken: 8.722 seconds
```

```
hive> insert into customers partition (load_date = '2023-03-10')values(3,"Albin",24,"Pune",50000);
Query ID = root_20230309085540_1bc1239b-2b0d-444f-bea5-e1c69576df93
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1678329811540_0008)

--------------------------------------------------------------------------------
      VERTICES       STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
--------------------------------------------------------------------------------
 Map 1 .......... SUCCEEDED      1        1       0       0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01 [==========================>>] 100% ELAPSED TIME: 3.34 s

Loading data to table studentdb.customers partition (load_date=2023-03-10)
Partition studentdb.customers{load_date=2023-03-10} stats: [numFiles=1, numRows=0, totalSize=863, rawDataSize=0]
OK
Time taken: 5.857 seconds
```

```
hive> select * from customers;
OK
1       Ross    25      Mumbai 25000 2023-03-09
2       Mike    27      Bhopal 35000 2023-03-09
3       Albin 24        Pune 50000 2023-03-10
Time taken: 0.302 seconds, Fetched: 3 row(s)
```

```
hive> create table orders(oid int, customer_id int, amount int);
OK
Time taken: 0.606 seconds
```

```
hive> insert into orders values(101, 2, 20000);
```

```
hive> insert into orders values(102, 2, 25000);
```

```
hive> insert into orders values(104,4,10000);
hive> select * from orders;
OK
101     2       20000
102     2       25000
103     1       30000
104     4       10000
Time taken: 0.174 seconds, Fetched: 4 row(s)
```

 # Executing Inner Join

```
hive> select * from customers c join orders o >
   on (c.id = o.customer_id);
Query ID = root_20230309090029_4b55b07e-4508-478f-a05b-4eaca94e4190
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id
application_1678329811540_0008)
 ----------------------------------------------------------------------
     VERTICES       STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
 ----------------------------------------------------------------------
Map 1 ..... SUCCEEDED     6        6      0      0      0      0
Map 2 ..... SUCCEEDED     1        1      0      0      0      0
 ----------------------------------------------------------------------
VERTICES: 02/02 [==========================>>] 100% ELAPSED TIME: 19.48 s
 ----------

OK
1       Ross 25     Mumbai 25000 2023-03-09         103       1          30000
2       Mike        27          Bhopal 35000 2023-03-09          101       2          20000
2       Mike    27        Bhopal 35000 2023-03-09       102     2       25000
Time taken: 20.319 seconds, Fetched: 3 row(s)
```

 # Executing LEFT OUTER JOIN

```
hive> select * from customers c left outer join orders o
    > on (c.id = o.customer_id);
Query ID = root_20230309090159_2d9c595a-7a7b-45d5-ad28-81e65ac22b64
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1678329811540_0008)
--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      6          6        0        0       0       0
Map 2 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 13.92 s
--------------------------------------------------------------------------------
OK
1       Ross    25      Mumbai  25000   2023-03-09      103     1       30000
2       Mike    27      Bhopal  35000   2023-03-09      101     2       20000
2       Mike    27      Bhopal 35000    2023-03-09      102     2       25000
3       Albin 24        Pune    50000 2023-03-10        NULL NULL NULL Time taken: 15.026 seconds,
        Fetched: 4 row(s)
```

```
select * from customers c right outer join orders o >
   on (c.id = o.customer_id);
Query ID = root_20230309090237_47fe2670-0826-4677-845e-b2cd58660148
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id
application_1678329811540_0008)
  ----------------------------------------------------------------------
      VERTICES       STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
  ----------------------------------------------------------------------
Map 1 ..... SUCCEEDED     6        6       0       0       0       0
Map 2 ..... SUCCEEDED     1        1       0       0       0       0
  ----------------------------------------------------------------------
VERTICES: 02/02 [==========================>>] 100% ELAPSED TIME: 10.82 s
  ----------

OK
2       Mike    27      Bhopal 35000 2023-03-09         101     2       20000
2       Mike    27      Bhopal 35000 2023-03-09         102     2       25000
1       Ross    25      Mumbai 25000 2023-03-09         103     1       30000
NULL    NULL    NULL    NULL    NULL    NULL    104         4       10000
Time taken: 11.516 seconds, Fetched: 4 row(s)
```

 # Executing RIGHT OUTER JOIN

 # Executing FULL OUTER JOIN

```
hive> select * from customers c full outer join orders o
   > on (c.id = o.customer_id);
Query ID = root_20230309090307_762521ad-0c69-4310-83f2-5d8061121d8a
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1678329811540_0008)
--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      6          6        0        0       2       0
Map 3 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 18.78 s
--------------------------------------------------------------------------------
OK
1       Ross    25      Mumbai  25000   2023-03-09      103     1       30000
2       Mike    27      Bhopal  35000   2023-03-09      101     2       20000
2       Mike    27      Bhopal  35000   2023-03-09      102     2       25000
3       Albin   24      Pune    50000   2023-03-10      NULL    NULL    NULL
NULL NULL NULL NULL NULL NULL 104               4       10000
Time taken: 19.841 seconds, Fetched: 5 row(s)
```

Views in HIVE

Views are generated based on user requirements. You can save any result set data as a view. The usage of view in Hive is same as that of the view in SQL. It is a standard RDBMS concept. We can execute all DML operations on a view.

Creating a View on customers table for address Mumbai.

```
hive> create view if not exists customers_vw
> as select * from customers where address="Mumbai";
OK
Time taken: 0.248 seconds
```

Displaying the results of above view

```
hive> select * from customers_vw;
OK
1       Ross    25      Mumbai 25000 2023-03-09 Time taken: 0.146 seconds, Fetched: 1 row(s)
```

Creating a view on complete customers table instead of one single row (condition based)

```
hive> alter view customers_vw as select * from customers;
OK
Time taken: 0.212 seconds
```

Displaying the results of above view

```
hive> select * from customers_vw;
OK
1       Ross   25    Mumbai 25000 2023-03-09
2       Mike   27    Bhopal 35000 2023-03-09
Time taken: 0.11 seconds, Fetched: 2 row(s)
```

Dropping the view created

```
hive> drop view if exists customers_vw;
OK
Time taken: 0.462 seconds hive> select * from customers_vw;
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'customers_vw'
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

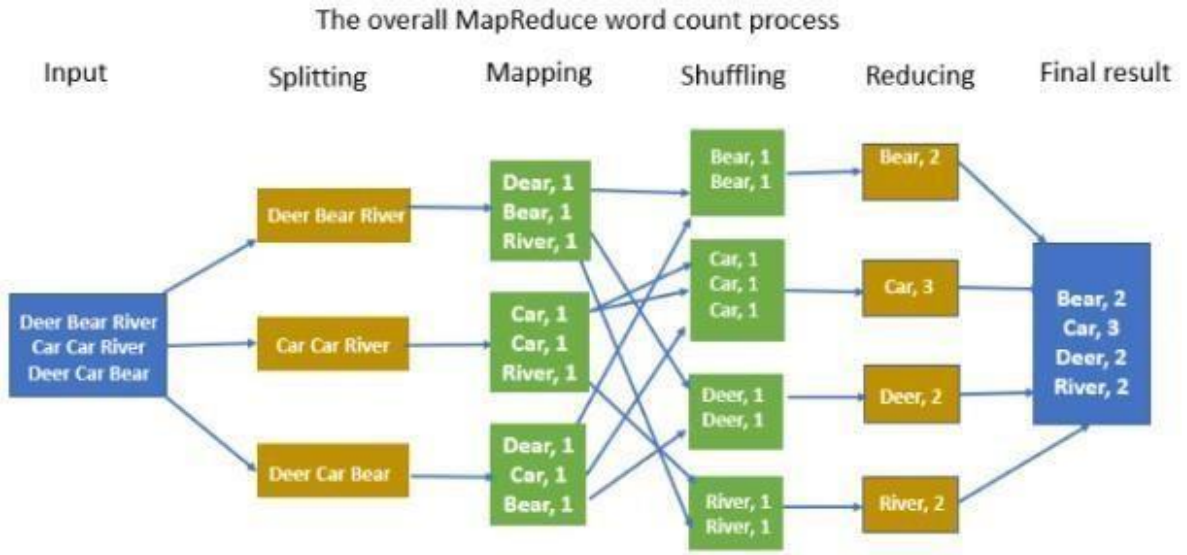
EXPERIMENT NO. 5
Map Reduce

AIM: Implement program for Word Count and Matrix Multiplication using Map Reduce.

THEORY:

Word Count

Hadoop can be developed in programming languages like Python and C++. MapReduce Hadoop is a software framework for ease in writing applications of software processing huge amounts of data. MapReduce Word Count is a framework which splits the chunk of data, sorts the map outputs and input to reduce tasks. A File-system stores the output and input of jobs. Re-execution of failed tasks, scheduling them and monitoring them is the task of the framework.



The input given is converted into the string. Then it tokenizes them into words as if it need to break them. The mapper will append a single number or digit to each word and mapper outputs are shown above. Once we get the outputs as key-value pairs, once we pass the offset address as input to the mapper, the output of the value would be key-value pairs.

The output is getting into the sorting and shuffling phase. When we sort based on keys, all the keys will come to once a particular place. Sorting on the keys and shuffling the keys is done. A single word will go to a single reducer. Input to the reducer is key-value pairs. Once we pass outputs to reducer as input, the reducer will sum up all the values to keys.

That is, it groups up all the similar keys and output would be the concatenated key-value pair. The reducer will pick the result from the temp path and it will arrive at the final result. When we execute map-reduce, the input and output should be created in HDFS.

CODE:

```python
from collections import Counter
from typing import List, Tuple
import itertools

def mapper(text: str) -> List[Tuple[str, int]]:
    words = text.split()
    return [(word, 1) for word in words]

def reducer(word: str, counts: List[int]) -> Tuple[str, int]:
    return (word, sum(counts))

def word_count_map_reduce(documents: List[str]) -> List[Tuple[str, int]]:
    # Map step
    mapped_values = itertools.chain.from_iterable(map(mapper, documents))

    # Shuffle and sort step (not necessary in this case)
    sorted_values = sorted(mapped_values, key=lambda x: x[0])

    # Reduce step
    reduced_values = [reducer(key, (val for _, val in pairs)) for key, pairs in itertools.groupby
        (sorted_values, key=lambda x: x[0])]

    return reduced_values


documents = [
    "deer bear river",
    "car car river",
    "deer car bear",
]

word_counts = word_count_map_reduce(documents)

print(word_counts)
```

OUTPUT:

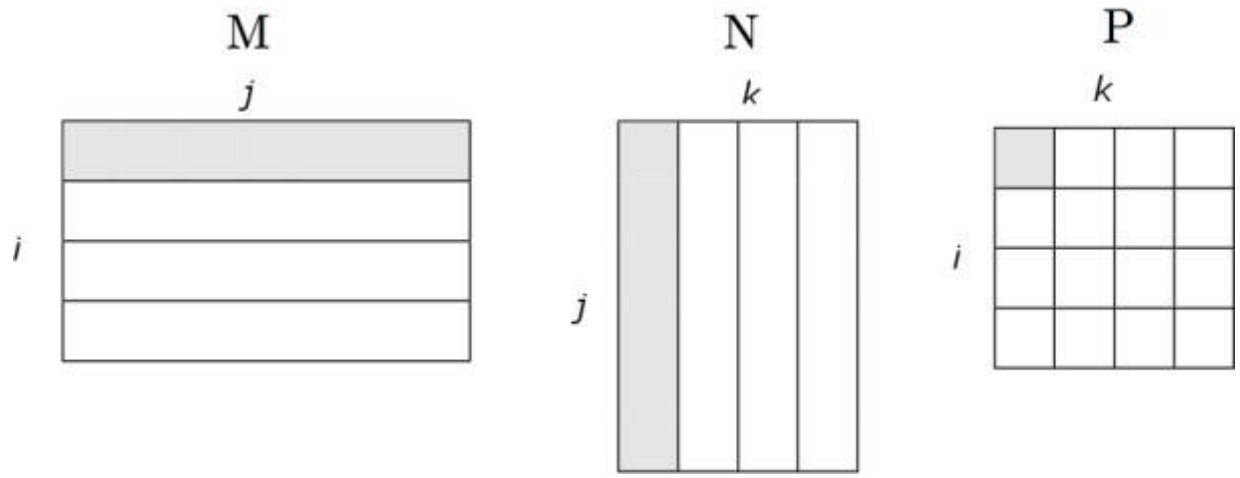
```
[('bear', 2), ('car', 3), ('deer', 2), ('river', 2)]
```

Matrix Multiplication

Matrix-vector and matrix-matrix calculations fit nicely into the MapReduce style of computing. In this post I will only examine matrix-matrix calculation as described in [1, ch.2].

Suppose we have a pxq matrix M, whose element in row *i* and column *j* will be denoted m_{ij} and a qxr matrix N whose element in row j and column k is donated by n_{jk} then the product P = MN will be pxr matrix P whose element in row i and column k will be donated by p_{ik}, where $P(i,k) = m_{ij} * n_{jk}$

.



Matrix Data Model for MapReduce

We represent matrix M as a relation $M(I, J, V)$, with tuples (i, j, m_{ij}), and matrix N as a relation $N(J, K, W)$, with tuples (j, k, n_{jk}). Most matrices are sparse so large amount of cells have value zero. When we represent matrices in this form, we do not need to keep entries for the cells that have values of zero to save large amount of disk space. As input data files, we store matrix M and N on HDFS in following format:

M, i, j, m_{ij}

M,0,0,10.0
M,0,2,9.0
M,0,3,9.0
M,1,0,1.0
M,1,1,3.0
M,1,2,18.0
M,1,3,25.2
....

N, j, k, n_{jk}

N,0,0,1.0
N,0,2,3.0
N,0,4,2.0
N,1,0,2.0
N,3,2,-1.0
N,3,6,4.0
N,4,6,5.0 N,4,0,-1.0
....

MapReduce

We will write Map and Reduce functions to process input files. Map function will produce $key, value$ pairs from the input data as it is described in Algorithm 1. Reduce function uses the output of the Map function and performs the calculations and produces $key, value$ pairs as described in Algorithm 2. All outputs are written to HDFS.

Algorithm 1: The Map Function

1 **for** *each element* m_{ij} *of* M **do**
2 produce $(key, value)$ pairs as $((i,k),(M,j,m_{ij}))$, for $k = 1,2,3,..$ up to the number of columns of N
3 **for** *each element* n_{jk} *of* N **do**
4 produce $(key, value)$ pairs as $((i,k),(N,j,n_{jk}))$, for $i = 1,2,3,...$ up to the number of rows of M
5 **return** *Set of (key, value) pairs that each key,* (i,k)*, has a list with values* (M,j,m_{ij}) *and* (N,j,n_{jk}) *for all possible values of* j

Algorithm 2: The Reduce Function

1 **for** *each key (i,k)* **do**
2 sort values begin with M by j in $list_M$
3 sort values begin with N by j in $list_N$
4 multiply m_{ij} and n_{jk} for j_{th} value of each list
5 sum up $m_{ij} * n_{jk}$
6 **return** $(i,k), \sum_{j=1} m_{ij} * n_{jk}$

The value in row *i* and column *k* of product matrix *P* will be:
$P_{(i,k)} = \sum_{j=1} m_{ij} * n_{jk}$.

Let me examine the algorithms on an example to explain the algorithms better. Suppose we have two matrices, *M*, 2×3 matrix, and *N*, 3×2 matrix as follows:

The product *P* of *MN* will be as follows:

$$\begin{bmatrix} 1a+2c+3e & 1b+2d+3f \\ 4a+5c+6e & 4b+5d+6f \end{bmatrix}$$

CODE:

```python
import itertools

def map_matrix_mult(args):
    row_a, col_b = args
    return ((row_a, col_b, sum(matrix_a[row_a][col_a] * matrix_b[row_b][col_b] for col_a, row_b in zip
        (range(len(matrix_a[0])), range(len(matrix_b))))) for col_b in range(len(matrix_b[0])))

def reduce_matrix_mult(key, values):
    row_idx, col_idx, val = key[0], key[1], sum(values)
    return row_idx, col_idx, val

# example matrices
matrix_a = [[1, 2, 3], [4, 5, 6]]
matrix_b = [[7, 8], [9, 10], [11, 12]]

# initialize result matrix with zeros
result_matrix = [[0 for col in range(len(matrix_b[0]))] for row in range(len(matrix_a))]

# calculate matrix multiplication
flattened_values = list(itertools.chain.from_iterable(map(map_matrix_mult, [(row_a, col_b) for row_a in
    range(len(matrix_a)) for col_b in range(len(matrix_b[0]))]) ))
reduced_values = [reduce_matrix_mult(key, (val for _, _, val in pairs)) for key, pairs in itertools
    .groupby(sorted(filter(lambda x: x is not None, flattened_values)), key=lambda x: (x[0], x[1], 0))]

for row_idx, col_idx, val in reduced_values:
    result_matrix[row_idx][col_idx] = val

# print the result matrix
print("Matrix A",matrix_a)
print("Matrix B",matrix_b)
print("Result Matrix",result_matrix)
```

OUTPUT:

```
Matrix A [[1, 2, 3], [4, 5, 6]]
Matrix B [[7, 8], [9, 10], [11, 12]]
Result Matrix [[116, 128], [278, 308]]
```

CONCLUSION: We have successfully implemented program for Word Count and Matrix Multiplication using MapReduce.

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 6
Installation and Configuration of Apache Spark

AIM: Install and Configure Apache Spark

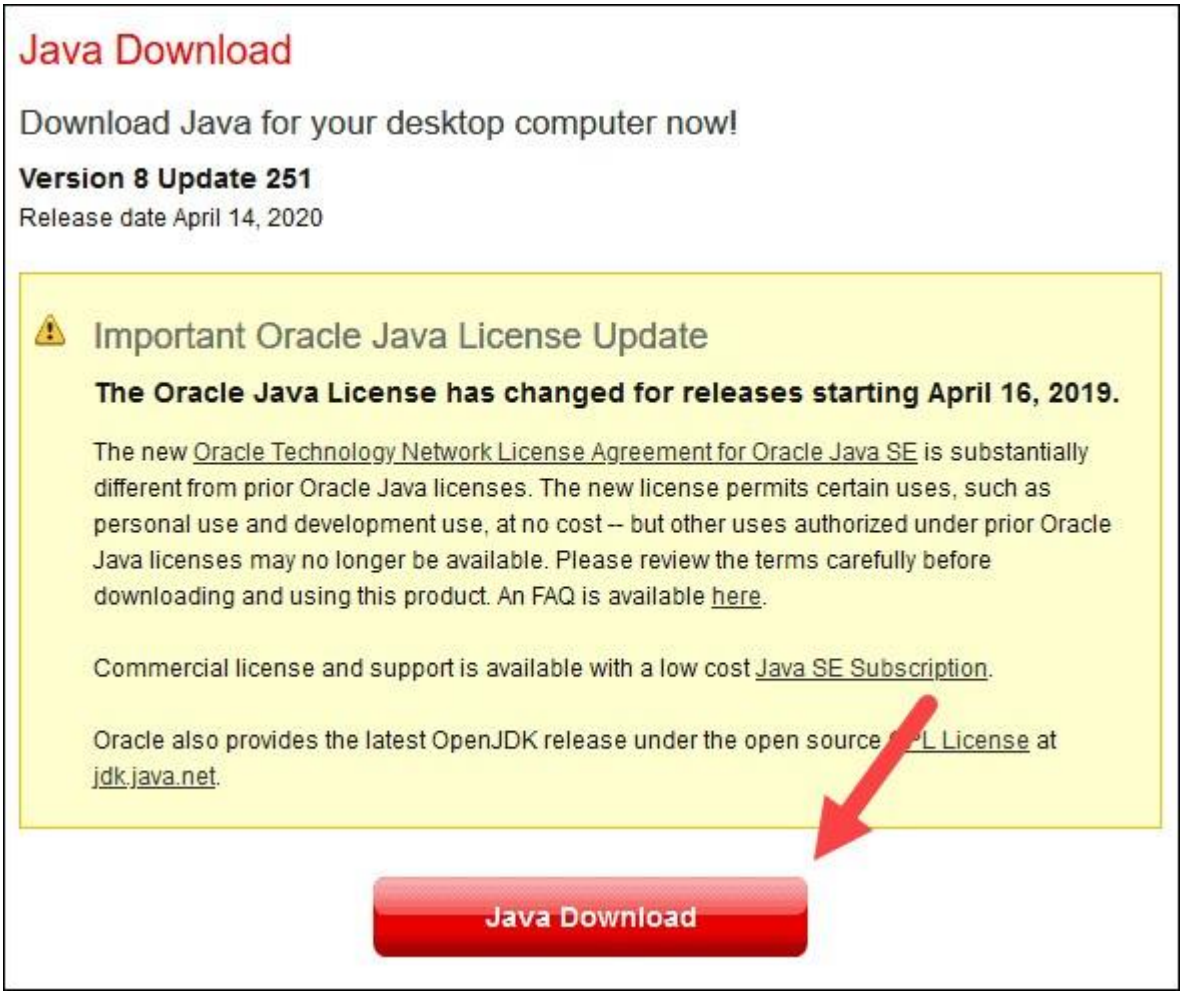
THEORY:
Apache Spark is an open-source data processing framework for large volumes of data from multiple sources. Spark is used in distributed computing for processing machine learning applications, data analytics, and graph-parallel processing on single-node machines or clusters.

Owing to its lightning-fast processing speed, scalability, and programmability for Big Data, Spark has become one of the most widely used Big Data distributed processing frameworks for scalable computing.

Thousands of companies, including tech giants like Apple, Facebook, IBM, and Microsoft, use Apache Spark. Spark Installation is simple and can be done in a variety of ways. It provides native bindings for programming languages, including Java, Scala, Python, and R.

Steps in Apache Spark Installation Step 1: Install Java 8
Apache Spark requires Java 8. You can check to see if Java is installed using the command prompt.
Open the command line by clicking Start > type *cmd* > click Command Prompt. Type
the following command in the command prompt:
java -version
If Java is installed, it will respond with the following output:

Your version may be different. The second digit is the Java version – in this case, Java 8. If
you don't have Java installed:
1. Open a browser window, and navigate to https://java.com/en/download/.

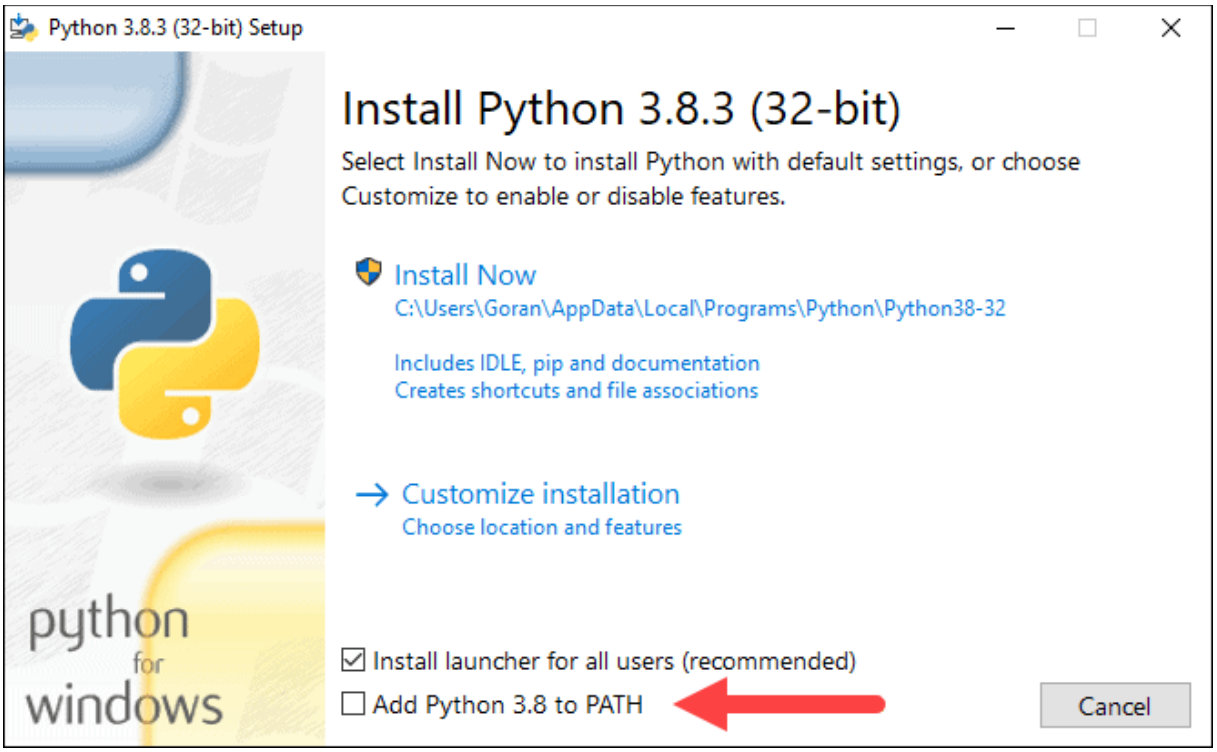


2. Click the Java Download button and save the file to a location of your choice.
3. Once the download finishes double-click the file to install Java.

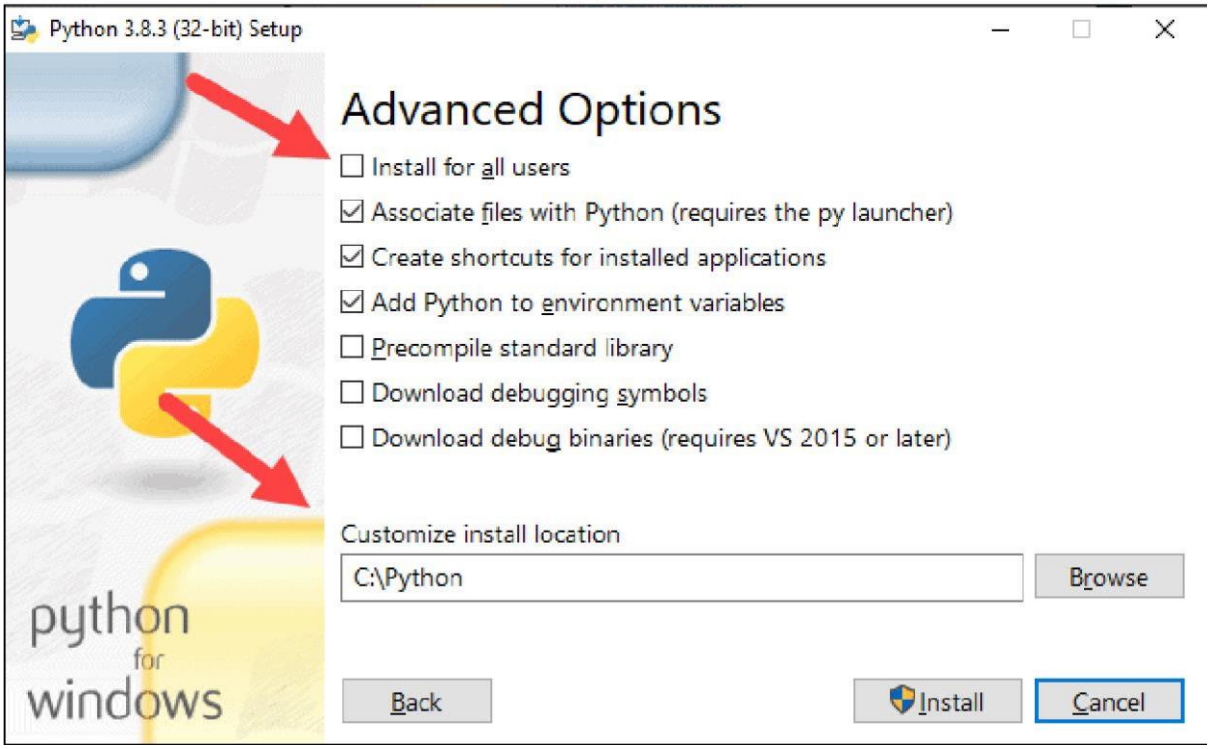
Step 2: Install Python

1. To install the Python package manager, navigate to https://www.python.org/ in your web browser.
2. Mouse over the Download menu option and click Python 3.8.3. 3.8.3 is the latest version at the time of writing the article.
3. Once the download finishes, run the file.

4. Near the bottom of the first setup dialog box, check off *Add Python 3.8 to PATH*. Leave the other box checked.
5. Next, click Customize installation.



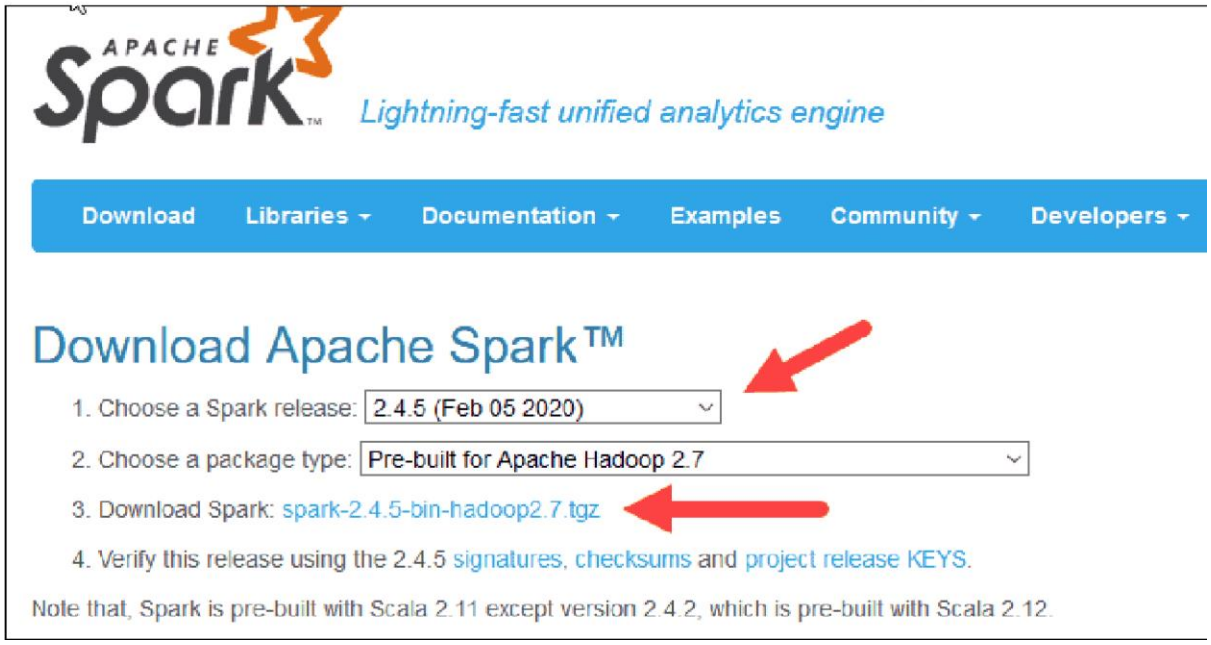
6. You can leave all boxes checked at this step, or you can uncheck the options you do not want.
7. Click Next.
8. Select the box Install for all users and leave other boxes as they are.
9. Under *Customize install location*, click Browse and navigate to the C drive. Add a new folder and name it *Python*.
10. Select that folder and click OK.



11. Click Install, and let the installation complete.
12. When the installation completes, click the *Disable path length limit* option at the bottom and then click Close.
13. If you have a command prompt open, restart it. Verify the installation by checking the version of Python: python --version The output should print Python 3.8.3.

Step 3: Download Apache Spark

1. Open a browser and navigate to https://spark.apache.org/downloads.html.
2. Under the *Download Apache Spark* heading, there are two drop-down menus. Use the current non-preview version.
   - In our case, in *Choose a Spark release* drop-down menu select 2.4.5 (Feb 05 2020).
   - In the second drop-down *Choose a package type*, leave the selection Pre-built for Apache Hadoop 2.7.
3. Click the *spark-2.4.5-bin-hadoop2.7.tgz* link.



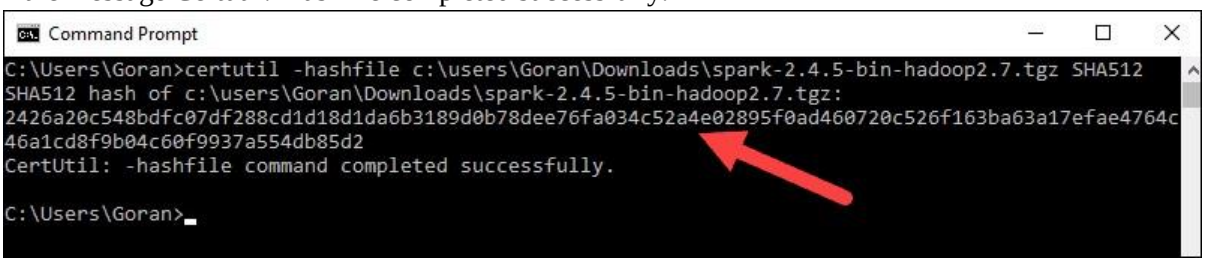
4. A page with a list of mirrors loads where you can see different servers to download from. Pick any from the list and save the file to your Downloads folder.

Step 4: Verify Spark Software File

1. Verify the integrity of your download by checking the checksum of the file. This ensures you are working with unaltered, uncorrupted software.
2. Navigate back to the *Spark Download* page and open the Checksum link, preferably in a new tab.
3. Next, open a command line and enter the following command:

certutil -hashfile c:\users\username\Downloads\spark-2.4.5-bin-hadoop2.7.tgz SHA512

4. Change the username to your username. The system displays a long alphanumeric code, along with the message Certutil: -hashfile completed successfully.



5. Compare the code to the one you opened in a new browser tab. If they match, your download file is uncorrupted.

Step 5: Install Apache Spark

Installing Apache Spark involves extracting the downloaded file to the desired location.

1. Create a new folder named *Spark* in the root of your C: drive. From a command line, enter the following:

```
cd \
mkdir Spark
```

2. In Explorer, locate the Spark file you downloaded.

3. Right-click the file and extract it to *C:\Spark* using the tool you have on your system (e.g., 7-Zip). 4. Now, your *C:\Spark* folder has a new folder *spark-2.4.5-bin-hadoop2.7* with the necessary files inside.

Step 6: Add winutils.exe File

Download the winutils.exe file for the underlying Hadoop version for the Spark installation you downloaded.

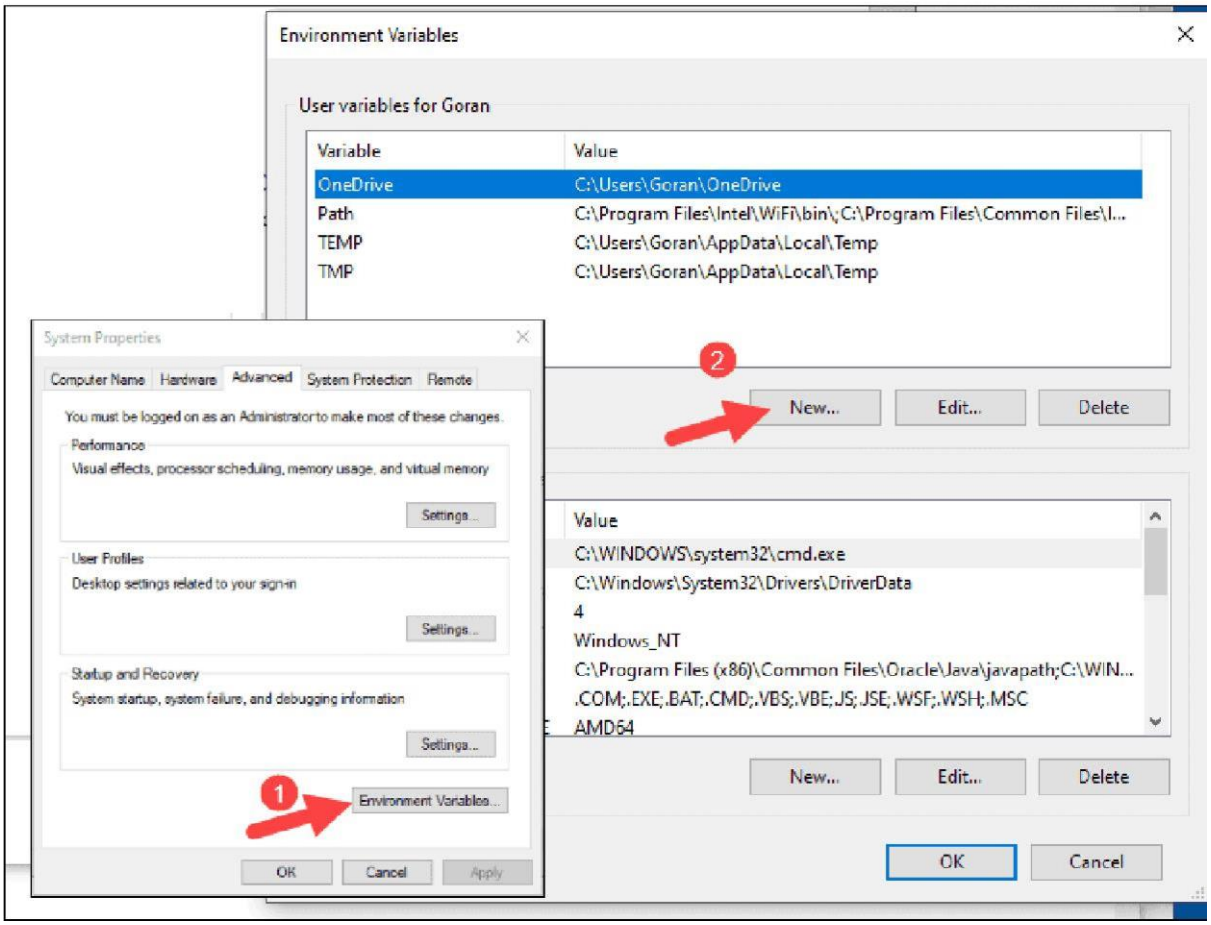
1. Navigate to this URL https://github.com/cdarlint/winutils and inside the bin folder, locate winutils.exe, and click it.

2. Find the Download button on the right side to download the file.
3. Now, create new folders *Hadoop* and bin on C: using Windows Explorer or the Command Prompt.
4. Copy the winutils.exe file from the Downloads folder to C:\hadoop\bin.

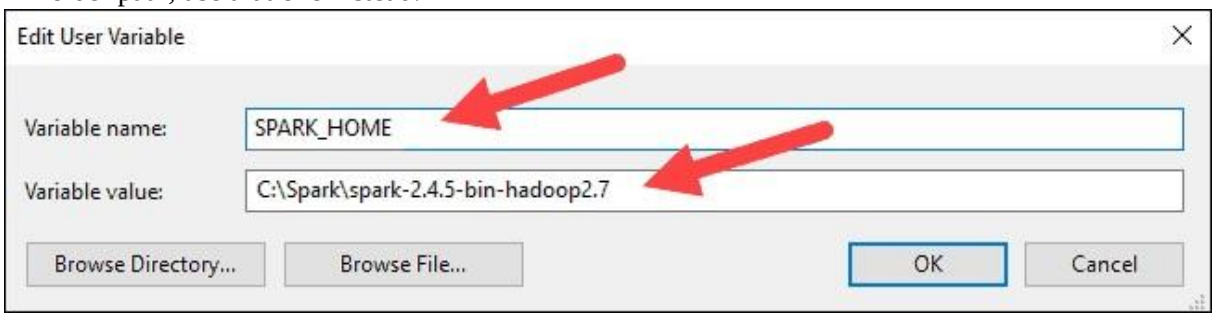
Step 7: Configure Environment Variables

Configuring environment variables in Windows adds the Spark and Hadoop locations to your system PATH. It allows you to run the Spark shell directly from a command prompt window.

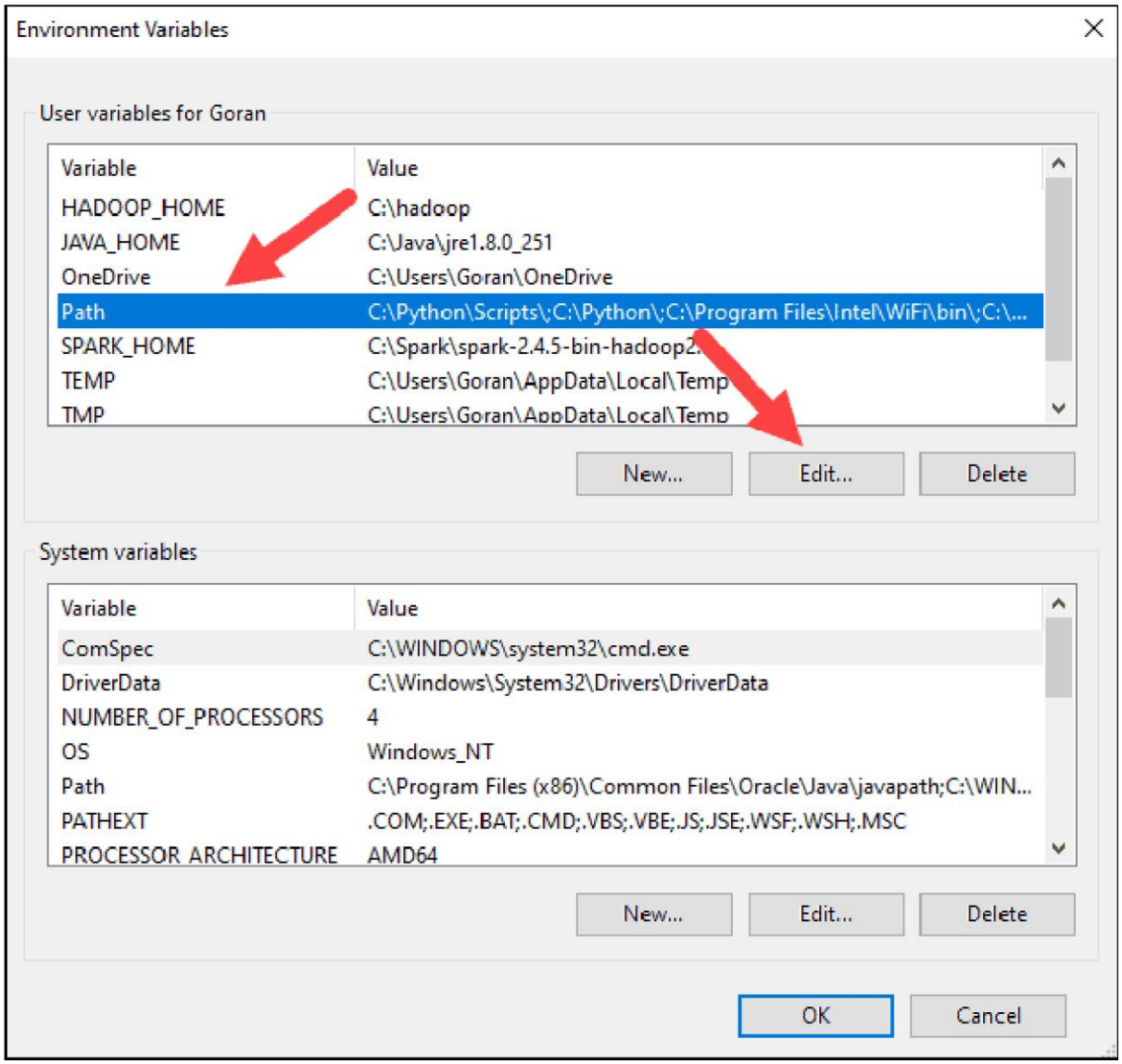
1. Click Start and type *environment*.
2. Select the result labeled *Edit the system environment variables*.
3. A System Properties dialog box appears. In the lower-right corner, click Environment Variables and then click New in the next window.



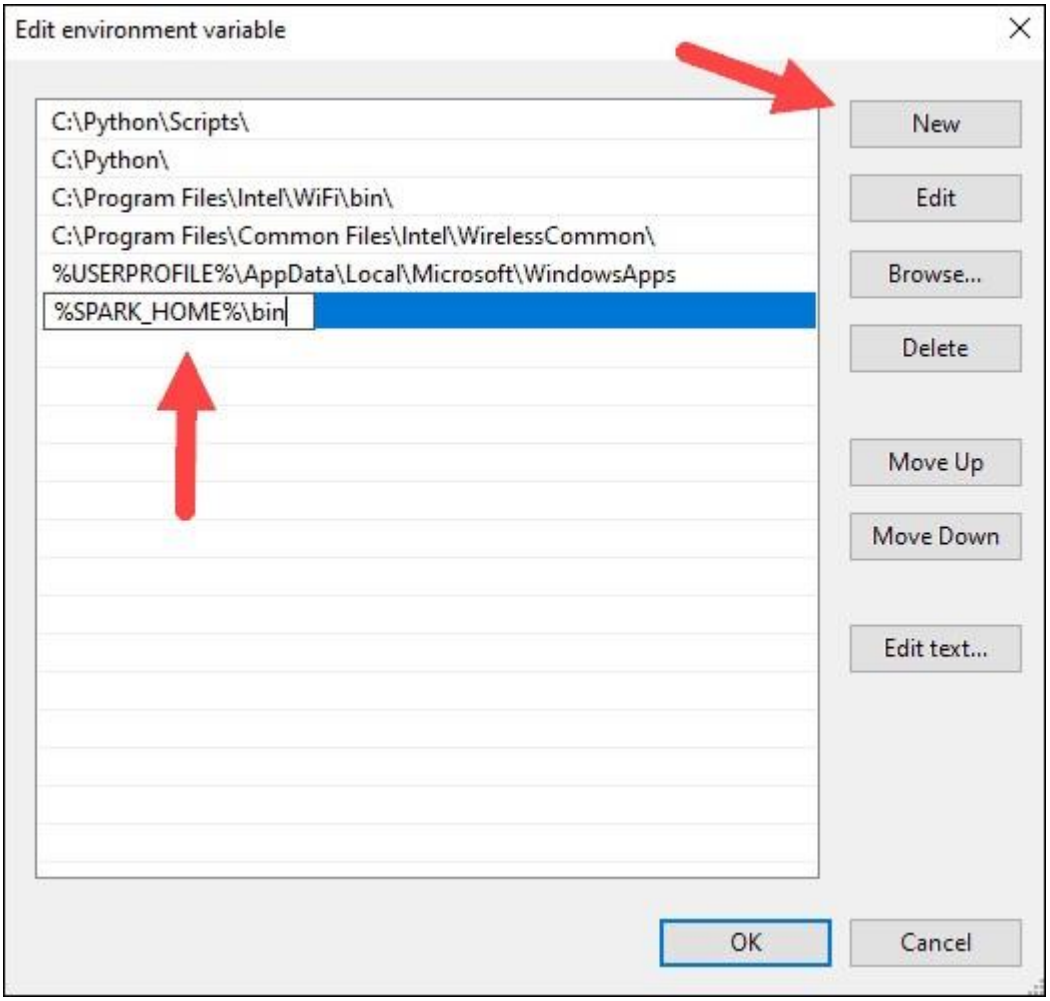
4. For *Variable Name* type *SPARK_HOME*.
5. For *Variable Value* type C:\Spark\spark-2.4.5-bin-hadoop2.7 and click OK. If you changed the folder path, use that one instead.



6. In the top box, click the Path entry, then click Edit. Be careful with editing the system path. Avoid deleting any entries already on the list.



7. You should see a box with entries on the left. On the right, click New.
8. The system highlights a new line. Enter the path to the Spark folder C:\Spark\spark-2.4.5-bin-hadoop2.7\bin. We recommend using %SPARK_HOME%\bin to avoid possible issues with the path.



9. Repeat this process for Hadoop and Java.
    - For Hadoop, the variable name is HADOOP_HOME and for the value use the path of the folder you created earlier: C:\hadoop. Add C:\hadoop\bin to the Path variable field, but we recommend using %HADOOP_HOME%\bin.
    - For Java, the variable name is JAVA_HOME and for the value use the path to your Java JDK directory (in our case it's C:\Program Files\Java\jdk1.8.0_251).
10. Click OK to close all open windows.

Note: Start by restarting the Command Prompt to apply changes. If that doesn't work, you will need to reboot the system.

Step 8: Launch Spark

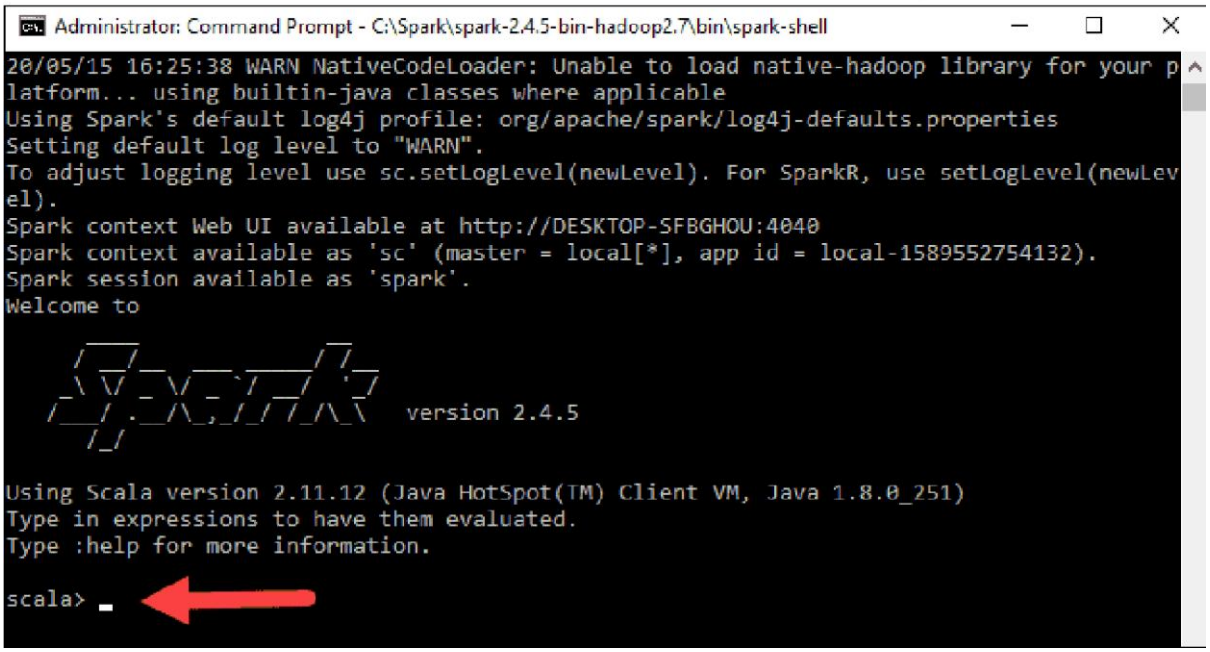
1. Open a new command-prompt window using the right-click and Run as administrator:
2. To start Spark, enter:

C:\Spark\spark-2.4.5-bin-hadoop2.7\bin\spark-shell

If you set the environment path correctly, you can type spark-shell to launch Spark.

3. The system should display several lines indicating the status of the application. You may get a Java pop-up. Select Allow access to continue.

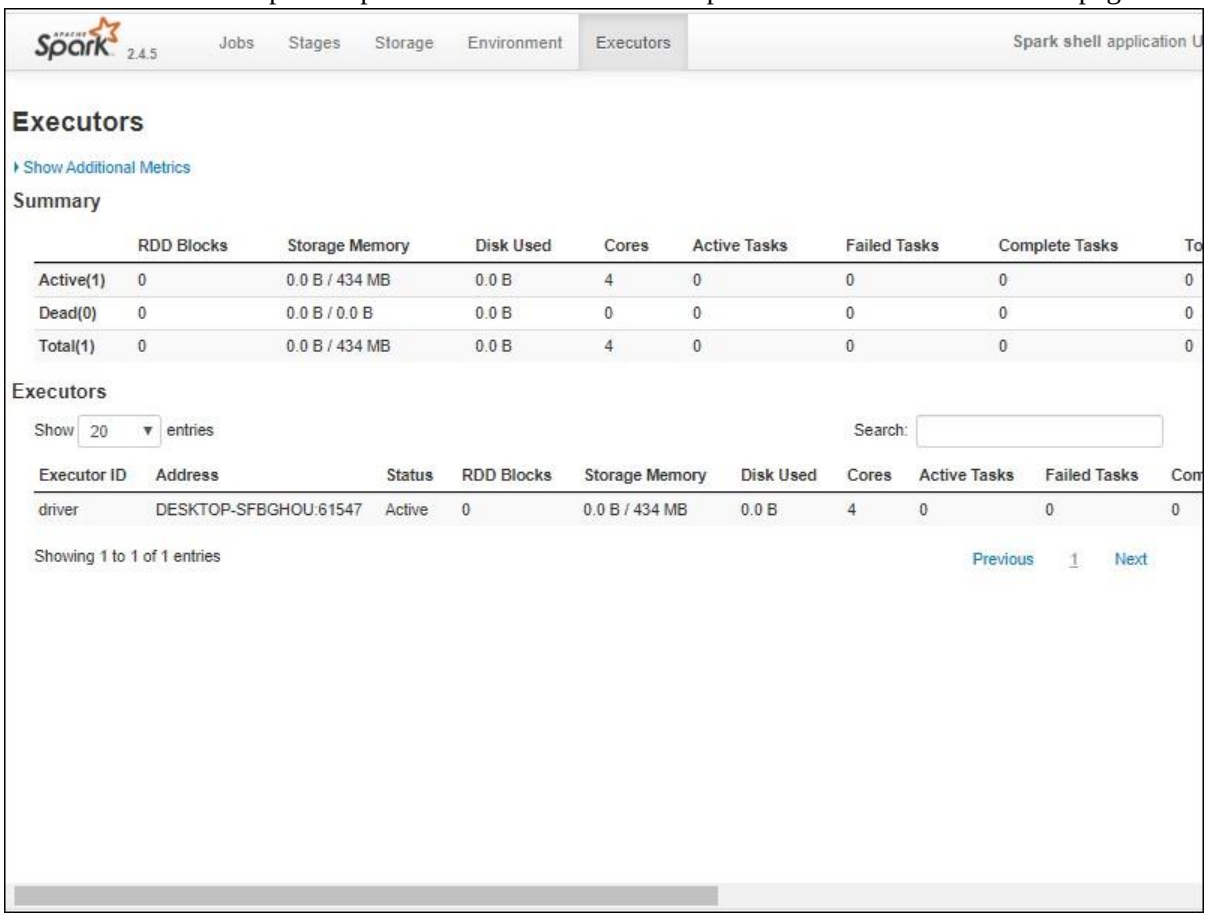
Finally, the Spark logo appears, and the prompt displays the Scala shell.



4., Open a web browser and navigate to http://localhost:4040/.

5. You can replace localhost with the name of your system.

6. You should see an Apache Spark shell Web UI. The example below shows the *Executors* page.



7. To exit Spark and close the Scala shell, press ctrl-d in the command-prompt window.

CONCLUSION: We have successfully installed and configured Apache Spark in Windows.

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 7
Execution of ML algorithms using Apache Spark MLlib

AIM: Implement and execute ML algorithms using Apche Spark MLlib.

THEORY:

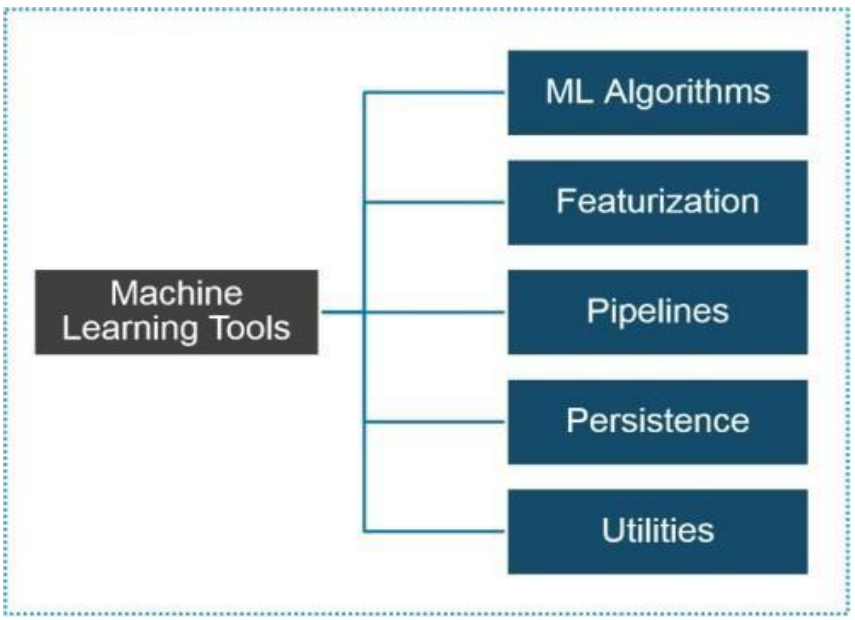
Spark MLlib is Apache Spark's Machine Learning component. MLlib consists of popular algorithms and utilities. MLlib in Spark is a scalable Machine learning library that discusses both high-quality algorithm and high speed. The machine learning algorithms like regression, classification, clustering, pattern mining, and collaborative filtering. Lower-level machine learning primitives like generic gradient descent optimization algorithm are also present in MLlib.

spark.mllib contains the original API built on top of RDDs. It is currently in maintenance mode.

spark.ml provides higher level API built on top of DataFrames for constructing ML pipelines. spark.ml is the primary Machine Learning API for Spark at the moment.

Spark MLlib Tools:

- **ML Algorithms:** ML Algorithms form the core of MLlib. These include common learning algorithms such as classification, regression, clustering and collaborative filtering.
- **Featurization:** Featurization includes feature extraction, transformation, dimensionality reduction and selection.
- **Pipelines:** Pipelines provide tools for constructing, evaluating and tuning ML Pipelines.
- **Persistence:** Persistence helps in saving and loading algorithms, models and Pipelines.
- **Utilities:** Utilities for linear algebra, statistics and data handling.



MLlib Algorithms:

The popular algorithms and utilities in Spark MLlib are:

- Basic Statistics
- Regression
- Classification
- Recommendation System
- Clustering
- Dimensionality Reduction
- Feature Extraction
- Optimization

Code:

```
# install pyspark
!pip install pyspark # initialise session
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.sql import SQLContext sc =
SparkContext('local') spark =
SparkSession(sc) sqlContext = SQLContext(sc)
```

1. Regeression

```
df=sqlContext.read.format('com.databricks.spark.csv').options(
header = 'true', inferschema= 'true').load('healthcaredataset-
stroke-data.csv') df.take(1) pd_df = df.toPandas()
print(pd_df) pd_df.dtypes df.cache() df.printSchema()
df.cache() df.printSchema() df.head() from pyspark.ml.feature
import VectorAssembler vectorAssembler =
VectorAssembler(inputCols = ['age', 'avg_glucose_level',
'hypertension', 'heart_disease',
'stroke'], outputCol = 'features') tdf =
vectorAssembler.transform(df) print(tdf)
tdf = tdf.select(['features', 'stroke'])
tdf.show(3) splits =
tdf.randomSplit([0.7, 0.3]) train_df =
splits[0] test_df = splits[1]
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(featuresCol = 'features',
labelCol='stroke', maxIter=10, regParam=0.3,
elasticNetParam=0.8) lr_model = lr.fit(train_df)
print("Coefficients: " + str(lr_model.coefficients))
print("Intercept: " + str(lr_model.intercept))
trainingSummary = lr_model.summary print("RMSE: %f" %
trainingSummary.rootMeanSquaredError) print("r2: %f" %
trainingSummary.r2)
```

2. Classification

```
df=sqlContext.read.format('com.databricks.spark.csv').options
( header = 'true', inferschema = 'true' ).load( 'salary.csv'
) df.take(1) df.cache() df.printSchema() df.dtypes from
pyspark.ml.feature import StringIndexer indexer0 =
```

```
StringIndexer(inputCol="workclass",
outputCol="workclassEncoded") indexed0 =
indexer0.fit(df).transform(df) indexer1 =
StringIndexer(inputCol="education",
outputCol="educationEncoded") indexed1 =
indexer1.fit(indexed0).transform(indexed0) indexer2 =
StringIndexer(inputCol="occupation",
outputCol="occupationEncoded") indexed2 =
indexer2.fit(indexed1).transform(indexed1) indexer3 =
StringIndexer(inputCol="sex", outputCol="sexEncoded")
indexed3 = indexer3.fit(indexed2).transform(indexed2)
indexer4 = StringIndexer(inputCol="native-country",
outputCol="countryEncoded") indexed4 =
indexer4.fit(indexed3).transform(indexed3) indexer5 =
StringIndexer(inputCol="salary", outputCol="salaryEncoded")
indexed5 = indexer5.fit(indexed4).transform(indexed4)
indexed5.show() from pyspark.ml.feature import
VectorAssembler vectorAssembler = VectorAssembler(inputCols =
['age', 'workclassEncoded', 'educationEncoded',
'occupationEncoded',
'sexEncoded', 'countryEncoded', 'education-num', 'capitalgain',
'hours-per-week', 'capital-loss'], outputCol =
'features') tdf = vectorAssembler.transform(indexed5)
print(tdf) tdf = tdf.select(['features',
'salaryEncoded']) tdf.show(3) train, test =
tdf.randomSplit([0.7, 0.3], seed = 2018)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import LinearSVC lsvc =
LinearSVC(featuresCol = 'features', labelCol =
'salaryEncoded', maxIter=10, regParam=0.1) lr =
LogisticRegression(featuresCol = 'features', labelCol =
'salaryEncoded', maxIter=10) lsvcModel =
lsvc.fit(train) lrModel = lr.fit(train)
print("Coefficients: " + str(lsvcModel.coefficients))
print("Intercept: " + str(lsvcModel.intercept))
import matplotlib.pyplot as plt trainingSummary =
lrModel.summary roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR']) plt.ylabel('False
Positive Rate') plt.xlabel('True Positive Rate')
plt.title('ROC Curve') plt.show() print('Training set
areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```

OUTPUT:

1. **Regression**

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	N/A	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	N/A	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

```
id                   int32
gender              object
age                float64
hypertension         int32
heart_disease        int32
ever_married        object
work_type           object
Residence_type      object
avg_glucose_level  float64
bmi                 object
smoking_status      object
stroke               int32
dtype: object
```

```
root
 |-- id: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- age: double (nullable = true)
 |-- hypertension: integer (nullable = true)
 |-- heart_disease: integer (nullable = true)
 |-- ever_married: string (nullable = true)
 |-- work_type: string (nullable = true)
 |-- Residence_type: string (nullable = true)
 |-- avg_glucose_level: double (nullable = true)
 |-- bmi: string (nullable = true)
 |-- smoking_status: string (nullable = true)
 |-- stroke: integer (nullable = true)
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
id	5110	36517.82935420744	21161.72162482715	67	72940
gender	5110	None	None	Female	Other
age	5110	43.226614481409015	22.61264672311348	0.08	82.0
hypertension	5110	0.0974559686888454	0.296606674233791	0	1
heart_disease	5110	0.05401174168297456	0.22606298750336554	0	1
ever_married	5110	None	None	No	Yes
work_type	5110	None	None	Govt_job	children
Residence_type	5110	None	None	Rural	Urban
avg_glucose_level	5110	106.14767710371804	45.28356015058193	55.12	271.74
bmi	5110	28.893236911794673	7.85406672968016	10.3	N/A
smoking_status	5110	None	None	Unknown	smokes
stroke	5110	0.0487279843444227	0.21531985698023753	0	1

```
Row(id=9046, gender='Male', age=67.0, hypertension=0, heart_disease=1, ever_married='Yes',
work_type='Private', Residence_type='Urban', avg_glucose_level=228.69, bmi='36.6',
smoking_status='formerly smoked', stroke=1)
```

```
[('id', 'int'),
 ('gender', 'string'),
 ('age', 'double'),
 ('hypertension', 'int'),
 ('heart_disease', 'int'),
 ('ever_married', 'string'),
 ('work_type', 'string'),
 ('Residence_type', 'string'),
 ('avg_glucose_level', 'double'),
 ('bmi', 'string'),
 ('smoking_status', 'string'),
 ('stroke', 'int')]
```



```
DataFrame[id: int, gender: string, age: double, hypertension: int, heart_disease: int,
ever_married: string, work_type: string, Residence_type: string, avg_glucose_level: double,
bmi: string, smoking_status: string, stroke: int, features: vector]
+--------------------+------+
|            features|stroke|
+--------------------+------+
|[67.0,228.69,0.0,...|     1|
|[61.0,202.21,0.0,...|     1|
|[80.0,105.92,0.0,...|     1|
+--------------------+------+
only showing top 3 rows
```

Final Predictions

```
Coefficients: [0.0,0.0,0.0,0.0,0.0]
Intercept: 0.048020219039595 62
```

```
RMSE: 0.213809
r2: -0.000000
```

2. Classification

```
[Row(age=39, workclass=' State-gov', fnlwgt=77516, education=' Bachelors',
education-num=13, marital-status=' Never-married', occupation=' Adm-clerical',
relationship=' Not-in-family', race=' White', sex=' Male', capital-gain=2174,
capital-loss=0, hours-per-week=40, native-country=' United-States', salary='
<=50K')]
```

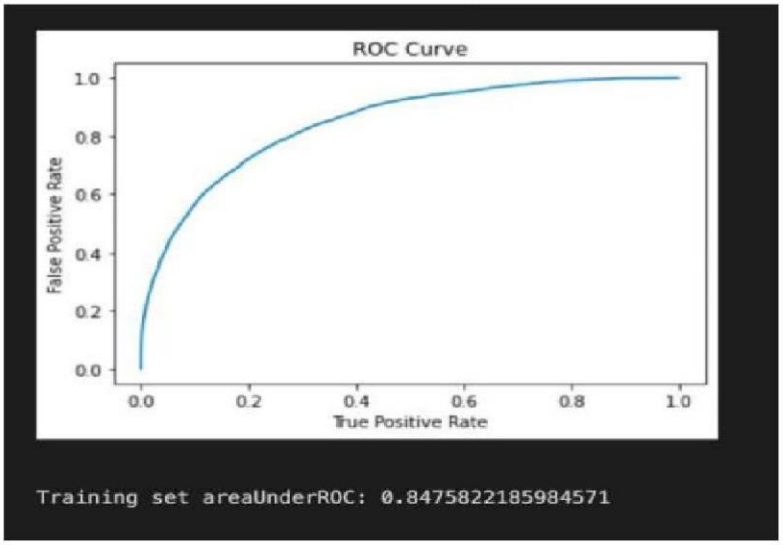
```
root
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: integer (nullable = true)
 |-- education: string (nullable = true)
 |-- education-num: integer (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital-gain: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- hours-per-week: integer (nullable = true)
 |-- native-country: string (nullable = true)
 |-- salary: string (nullable = true)
```

```
+---+----------------+------+------------+-------------+------------------+-----------------+--------------+------+------+------------+------------+--------------
+--------------+------+-----------------+----------------+------------------+----------+-------------+-------------+
|age|       workclass|fnlwgt|   education|education-num|    marital-status|       occupation|  relationship|  race|   sex|capital-gain|capital-loss|hours-per-
week|native-country|salary|workclassEncoded|educationEncoded|occupationEncoded|sexEncoded|countryEncoded|salaryEncoded|
+---+----------------+------+------------+-------------+------------------+-----------------+--------------+------+------+------------+------------+--------------
+--------------+------+-----------------+----------------+------------------+----------+-------------+-------------+
| 39|       State-gov| 77516|   Bachelors|           13|     Never-married|     Adm-clerical| Not-in-family| White|  Male|        2174|           0|            40|
United-States| <=50K|     4.0|      2.0|       3.0|   0.0|      0.0|      0.0|
| 50|Self-emp-not-inc| 83311|   Bachelors|           13|Married-civ-spouse|  Exec-managerial|       Husband| White|  Male|           0|           0|            13|
United-States| <=50K|     1.0|      2.0|       2.0|   0.0|      0.0|      0.0|
| 38|         Private|215646|     HS-grad|            9|          Divorced|Handlers-cleaners| Not-in-family| White|  Male|           0|           0|            40|
United-States| <=50K|     0.0|      0.0|       9.0|   0.0|      0.0|      0.0|
| 53|         Private|234721|        11th|            7|Married-civ-spouse|Handlers-cleaners|       Husband| Black|  Male|           0|           0|            40|
United-States| <=50K|     0.0|      5.0|       9.0|   0.0|      0.0|      0.0|
| 28|         Private|338409|   Bachelors|           13|Married-civ-spouse|   Prof-specialty|          Wife| Black|Female|           0|           0|            40|
Cuba| <=50K|      0.0|       2.0|     0.0|    1.0|      9.0|      0.0|
| 37|         Private|284582|     Masters|           14|Married-civ-spouse|  Exec-managerial|          Wife| White|Female|           0|           0|            40|
United-States| <=50K|     0.0|      3.0|       2.0|   1.0|      0.0|      0.0|
| 49|         Private|160187|         9th|            5|Married-spouse-a...|    Other-service| Not-in-family| Black|Female|           0|           0|            16|
Jamaica| <=50K|      0.0|       10.0|     5.0|    1.0|      11.0|      0.0|
```

```
DataFrame[age: int, workclass: string, fnlwgt: int, education: string, education-num: int,
marital-status: string, occupation: string, relationship: string, race: string, sex: string,
capital-gain: int, capital-loss: int, hours-per-week: int, native-country: string, salary:
string, workclassEncoded: double, educationEncoded: double, occupationEncoded: double,
sexEncoded: double, countryEncoded: double, salaryEncoded: double, features: vector]
+--------------------+-------------+
|            features|salaryEncoded|
+--------------------+-------------+
|[39.0,4.0,2.0,3.0...|          0.0|
|[50.0,1.0,2.0,2.0...|          0.0|
|(10,[0,3,6,8],[38...|          0.0|
+--------------------+-------------+
only showing top 3 rows
```

Final Predictions

```
Coefficients:
[0.009679140612290072,0.01342892012614891,0.02288921743340083,-0.02078106627747353,-0.2950315
05,0.008667393334614108,0.0003678549301454241]
Intercept: -2.5735276509983613
```




```
Training set areaUnderROC: 0.8475822185984571
```

CONCLUSION: Hence, with the help of the above experiment, we have successfully executed two ML algorithms using Apache Spark MLlib and compared its accuracy.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 8
MongoDB

AIM: Perform CRUD Operations using MongoDB.

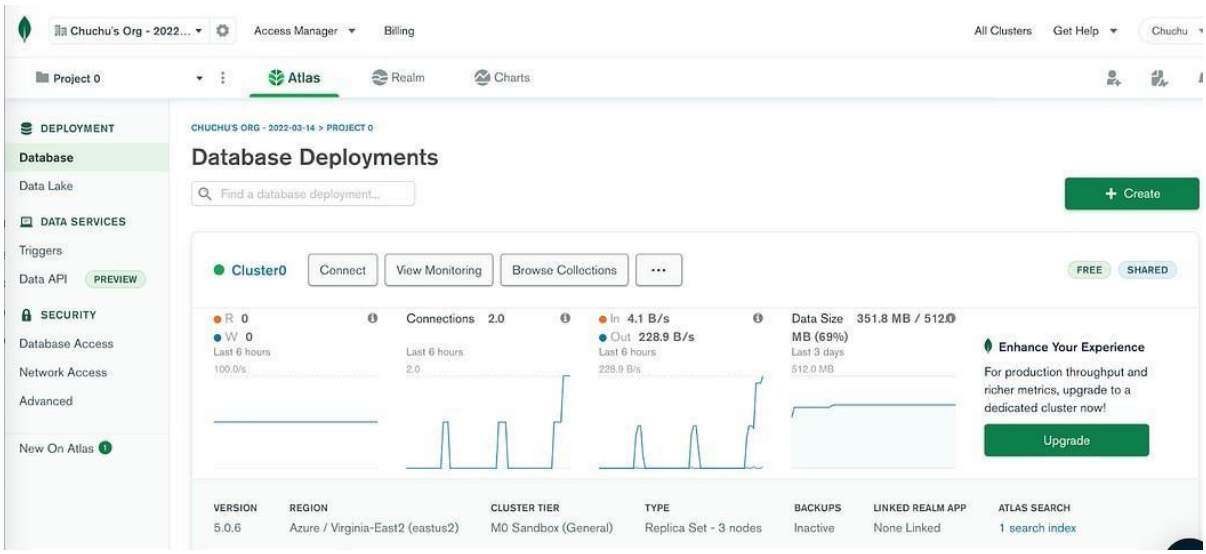
THEORY:

MongoDB is a document database that has a flexible schema for storing necessary data. A document is a record where the data is stored as key-value pairs in MongoDB. When retrieving information from the database, it can be in a JSON format while initially, each record is a document in BSON format. A JSON is a data format that can be used to store and retrieve data which is stored as key-value pairs. BSON is the binary representation of the data which is just a binary JSON.

MongoDB ensures scalability. It makes it easy to store data such that programmers have an easier time working with it because of its flexible schema. It is built to scale up quickly. It is easy to store unstructured and structured data because of the JSON like format that is used to store documents. MongoDB can handle high volumes of data which is needed having a large database of movies that can be streamed to the user in a movie streaming scenario.

Using mongo compass:

MongoDB Compass is a good GUI tool for managing mongoDB collections and doing data queries. It is free to use and can be run on macOS, Windows, and Linux.



A sample data set was created and used in order to test the functionality of the MongoDB database.

1. To show databases present in MongoDB Atlas. **`show dbs;`**

```
BDA          80.00 KiB
BDA1          8.00 KiB
admin       120.00 KiB
config       60.00 KiB
```

```
local          160.00 KiB
myDB             8.00 KiB


  poulations 56.00 KiB

test           80.00 KiB
```

2. **To select the database. use admin;**

'**switched to db admin**'

3. **To show all collections present in database.**

```
show collections;
employees Student
system.version
```

4. **To create a new collection.**

```
db.createCollection('Movie');
```

{ ok: 1 }

5. **To insert a single row in collection**.

```
db.Movie.insertOne({ID:1,Title:"Terminator",Director:"Sanjay",Year:1985,
Bud get:25000000});
```

{ acknowledged: true,
insertedId: ObjectId("**64302a42c27158d639b83646**") }

```
db.Movie.insertOne({ID:2,Title:"DDLJ",Director:"Karan",Year:1989,Budget:
200
00000});
```

{ acknowledged: true,
insertedId: ObjectId("**64302a86c27158d639b83647**") }

6. **To display values in collection.**

```
db.Movie.find({ID:2});
```

{ _id: ObjectId("**64302a86c27158d639b83647**"),

ID: 2,
Title: '**DDLJ**',
Director: '**Karan**',
Year: 1989,
Budget: 20000000 }

```
db.Movie.find();
```

{ _id: ObjectId("**64302a42c27158d639b83646**"),
ID: 1,
Title: '**Terminator**',
Director: '**Sanjay**',
Year: 1985,
Budget: 25000000 }
{ _id: ObjectId("**64302a86c27158d639b83647**"),
ID: 2,
Title: '**DDLJ**',

Director: '**Karan**',
Year: 1989,
Budget: 20000000 }

```
7. To insert multiple rows in collection.
db.Movie.insertMany([{ID:3,Title:"Dabang",Director:"salman",Year:2012,Bu
dge
t:30000000},{ID:4,Title:"Bagban",Director:"Dharmendra",Year:2010,Budget:
150 00000}]);
```

{ acknowledged: true, insertedIds:
{ '**0**': ObjectId("**64302b79c27158d639b83648**"), '**1**':
ObjectId("**64302b79c27158d639b83649**") } }

```
8. To display values in collection. db.Movie.find();
```

{ _id: ObjectId("**64302a42c27158d639b83646**"),

ID: 1,
Title: '**Terminator**',
Director: '**Sanjay**',
Year: 1985,
Budget: 25000000 }
{ _id: ObjectId("**64302a86c27158d639b83647**"),
ID: 2,
Title: '**DDLJ**', Director:
'**Karan**',

Year: 1989,
Budget: 20000000 }
{ _id: ObjectId("**64302b79c27158d639b83648**"),
ID: 3,
Title: '**Dabang**',
Director: '**salman**',
Year: 2012,
Budget: 30000000 }
{ _id: ObjectId("**64302b79c27158d639b83649**"),
ID: 4,
Title: '**Bagban**',
Director: '**Dharmendra**',
Year: 2010,
Budget: 15000000 }

```
8. To update value in the collection.
db.Movie.updateOne({'Title':'Terminator'},{$set:{'Title':'Robot'
}}); { acknowledged: true,
```

insertedId: null, matchedCount:
1, modifiedCount: 1,
upsertedCount: 0 }

```
db.Movie.find();
```

{ _id: ObjectId("**64302a42c27158d639b83646**"), ID:
1,
Title: '**Robot**', Director:
'**Sanjay**',
Year: 1985,
Budget: 25000000 }
{ _id: ObjectId("**64302a86c27158d639b83647**"),
ID: 2,
Title: '**DDLJ**',
Director: '**Karan**',
Year: 1989,
Budget: 20000000 }
{ _id: ObjectId("**64302b79c27158d639b83648**"),
ID: 3,
Title: '**Dabang**',
Director: '**salman**',
Year: 2012,
Budget: 30000000 }
{ _id: ObjectId("**64302b79c27158d639b83649**"),
ID: 4,
Title: '**Bagban**',
Director: '**Dharmendra**',
Year: 2010,
Budget: 15000000 }

```
db.Movie.find({$and:[{Year:{$lt:"2015-01-01"}},{Budget: {$gt:500000}}]})
```

9. To find maximum budget movies group by year.

```
db.Movie.aggregate([{$group : {_id : "$Year", num_tutorial : {$max :
"$Budget"}}}])
```

{ _id: 1985, num_tutorial: 25000000 }

{ _id: 1989, num_tutorial: 20000000 }

{ _id: 2010, num_tutorial: 15000000 }

{ _id: 2012, num_tutorial: 30000000 }

CONCLUSION: We have successfully executed CRUD operations in MongoDB Atlas.

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

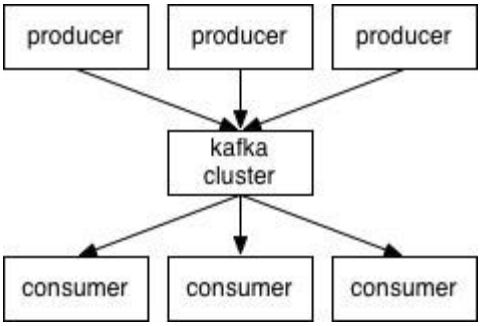
SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 9
Read streaming data using Kafka

AIM: Read streaming data using Kafka.

THEORY:
What is Apache Kafka?

Apache Kafka is a software platform which is based on a distributed streaming process. It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by LinkedIn, and later it was donated to the Apache Software Foundation. Currently, it is maintained by Confluent under Apache Software Foundation. Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.



What is a messaging system?

A messaging system is a simple exchange of messages between two or more persons, devices, etc. A publish-subscribe messaging system allows a sender to send/write the message and a receiver to read that message. In Apache Kafka, a sender is known as a producer who publishes messages, and a receiver is known as a consumer who consumes that message by subscribing it.

What is Streaming process?

A streaming process is the processing of data in parallelly connected systems. This process allows different applications to limit the parallel execution of the data, where one record executes without waiting for the output of the previous record. Therefore, a distributed streaming platform enables the user to simplify the task of the streaming process and parallel execution. Therefore, a streaming platform in Kafka has the following key capabilities:

1. As soon as the streams of records occur, it processes it.
2. It works similar to an enterprise messaging system where it publishes and subscribes streams of records.
3. It stores the streams of records in a fault-tolerant durable way.

To learn and understand Apache Kafka, the aspirants should know the following four core APIs:

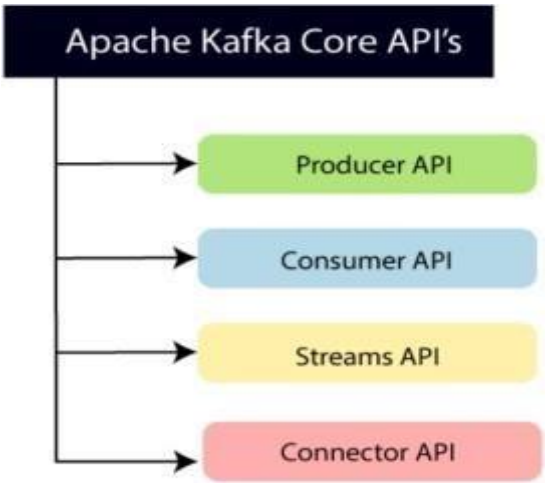
Producer API: This API allows/permits an application to publish streams of records to one or more topics.

Consumer API: This API allows an application to subscribe one or more topics and process the stream of records produced to them.

Streams API: This API allows an application to effectively transform the input streams to the output streams. It permits an application to act as a stream processor which consumes an input stream from one or more topics, and produce an output stream to one or more output topics.

Connector API: This API executes the reusable producer and consumer APIs with the existing data systems or applications.

Messages are published to topics by Kafka Producers. Those who subscribe to them, on the other hand, are called Kafka Consumers. Communication to and from Apache Kafka from Producers and Consumers is through a language agnostic TCP protocol, which is high-performing and simple. It assumes the responsibility for facilitating communications involving servers and clients.



Topics and Logs

A topic is considered a key abstraction in Kafka. A topic can be considered a feed name or category where the messages will be published. Each topic is further subdivided into partitions. Partitions split data across different nodes by Kafka brokers. In each of the messages within the partition, there is an ID number assigned, which is technically known as the offset.

Kafka Topic Partitions

All of the messages published can be retained in the cluster, regardless if they have been consumed or not. For instance, if the configuration states that it will be available only for one week, then it can be retained within such period only. Once the period has lapsed, it will be automatically deleted, which, in turn, will provide the system with additional space. On a per consumer basis, only the meta-data, which is also technically known as the offset, is going to be retained. The consumers will have complete control of this retention. It is often consumed linearly, but the user can often go back and reprocess, basically because he or she is in control. Consumers of Kafka are lightweight, cheap, and they do not have a significant impact on the way the clusters will perform. This differs from more traditional message systems. Every consumer has the ability to maintain their own offset, and hence, their actions will not affect other consumers.

Kafka Partitions

Partitions are leveraged for two purposes. The first is to adjust the size of the topic to make it fit on a single node. The second purpose of partition is for parallelism or performance tuning. It makes it possible for one consumer to simultaneously browse messages in concurrent threads.

Distribution

Every partition is replicated for fault tolerance. In every partition, there is a single server, which is labeled as the "Leader". The leader is responsible for the handling of the requests for read-write. There will also be "Followers", which can be zero or more nodes in the Kafka cluster. They will be responsible for replicating the actions undertaken by the leader. In the case of the failure of the leader, one of the followers will immediately assume the leadership position.

So, Why Kafka? When?

When compared to the conventional messaging system, one of the benefits of Kafka is that it has ordering guarantees. In the case of a traditional queue, messages are kept on the server based on the order at which they are kept. Then messages are pushed out based on the order of their storage and there is no specific timing requirement in their transmission. This means that their arrival to different consumers could be random. Hold the phone. Did you just write "could be random"? Think about that for 5 seconds. That may be fine in some use cases, but not others. Kafka is often deployed as the foundational element in Streaming Architectures because it can facilitate loose coupling between various components in architectures such as databases, microservices, Hadoop, data warehouses, distributed files systems, search applications, etc.

Guarantees

In a Kafka messaging system, there are essentially three guarantees:

1. A message sent by a producer to a topic partition is appended to the Kafka log based on the order sent.
2. Consumers see messages in the order of their log storage.
3. For topics with replication factor N, Kafka will tolerate N-1 number of failures without losing the messages previously committed to the log.

CODE

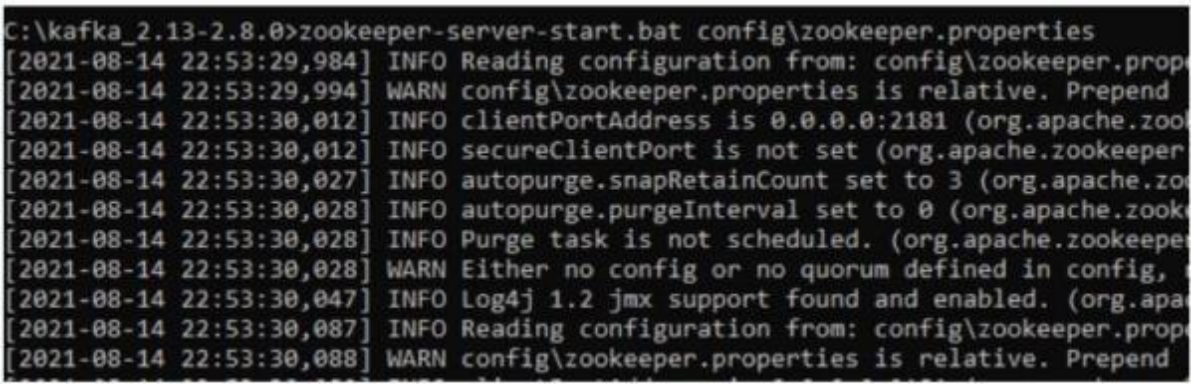
```
import findspark
findspark.init('')
import pyspark
from pyspark import RDD
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

if name=="main":
    sc = SparkContext(appName="PythonSparkStreamingKafka")
    ssc = StreamingContext(sc,60)
    #Creating Kafka direct stream
    message= KafkaUtils.createDirectStream(ssc, ["testtopic"],
{"metadata.broker.list":"|replace with your Kafka private
address|:9092"})
    words=message.map(lambda x:x[1].flatmap(lambda x:x.split(" ")))
    wordcount=words.map(lambda x: (x,1).reduceByKey(lambda a,b:a+b))
    wordcount.ppr int()
    #Starting Spark context
    ssc.start()
    ssc.awaitTermination()
```

Commands:

Open 4 command line terminals On first terminal enter the

command: **zookeeper-server-start.bat**

.\config\zookeeper.properties Output:

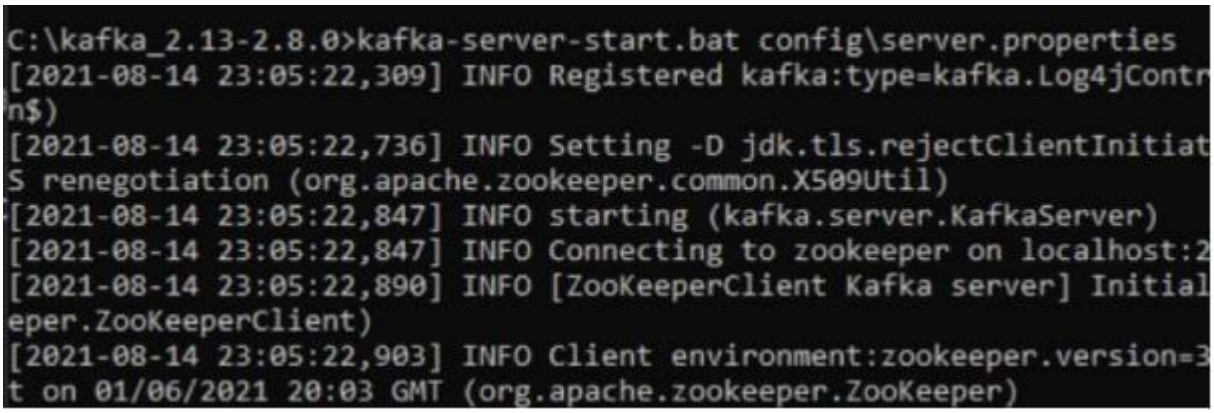


Zookeeper is up and running at the specified port. On

second terminal enter the command:

kafka-server-start.bat .\config\server.properties

Output:



Now, you have the kafka running in one command line window and zookeeper running in another. If you want to start working with them, you will have to open another command line window and start writing commands.

On third terminal we run the command: **kafka-console-producer.bat --**

broker-list localhost:9092 --topic testtopic

On fourth terminal we run the command:

```
C:\Users\ASUS DASH\Desktop\sem 6\bdi>spark-submit --packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.1.1
 kafkademo.py
```

```
Ivy Default Cache set to: /home/kafka/.ivy2/cache
The jars for the packages stored in: /home/kafka/.ivy2/jars
:: loading settings :: url = jar:file:/usr/local/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.apache.spark#spark-streaming-kafka-0-8_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
        confs: [default]
        found org.apache.spark#spark-streaming-kafka-0-8_2.11;2.1.1 in central
        found org.apache.kafka#kafka_2.11;0.8.2.1 in central
        found org.scala-lang.modules#scala-xml_2.11;1.0.2 in central
        found com.yammer.metrics#metrics-core;2.2.0 in central
        found org.slf4j#slf4j-api;1.7.16 in central
        found org.scala-lang.modules#scala-parser-combinators_2.11;1.0.2 in central
        found com.101tec#zkclient;0.3 in central
        found log4j#log4j;1.2.17 in central
        found org.apache.kafka#kafka-clients;0.8.2.1 in central
        found net.jpountz.lz4#lz4;1.3.0 in central
        found org.xerial.snappy#snappy-java;1.1.2.6 in central
        found org.apache.spark#spark-tags_2.11;2.1.1 in central
        found org.spark-project.spark#unused;1.0.0 in central
downloading https://repo1.maven.org/maven2/org/apache/spark/spark-streaming-kafka-0-8_2.11/2.1.1/spark-streaming-kafka-0-8_2.11-2.1.1.jar ...
        [SUCCESSFUL ] org.apache.spark#spark-streaming-kafka-0-8_2.11;2.1.1!spark-streaming-kafka-0-8_2.11.jar (693ms)
downloading https://repo1.maven.org/maven2/org/apache/spark/spark-tags_2.11/2.1.1/spark-tags_2.11-2.1.1.jar ...
        [SUCCESSFUL ] org.apache.spark#spark-tags_2.11;2.1.1!spark-tags_2.11.jar (229ms)
:: resolution report :: resolve 7138ms :: artifacts dl 965ms
        :: modules in use:
        com.101tec#zkclient;0.3 from central in [default]
        com.yammer.metrics#metrics-core;2.2.0 from central in [default]
        log4j#log4j;1.2.17 from central in [default]
        net.jpountz.lz4#lz4;1.3.0 from central in [default]
        org.apache.kafka#kafka-clients;0.8.2.1 from central in [default]
        org.apache.kafka#kafka_2.11;0.8.2.1 from central in [default]
        org.apache.spark#spark-streaming-kafka-0-8_2.11;2.1.1 from central in [default]
        org.apache.spark#spark-tags_2.11;2.1.1 from central in [default]
        org.scala-lang.modules#scala-parser-combinators_2.11;1.0.2 from central in [default]
        org.scala-lang.modules#scala-xml_2.11;1.0.2 from central in [default]
        org.slf4j#slf4j-api;1.7.16 from central in [default]
        org.spark-project.spark#unused;1.0.0 from central in [default]
        org.xerial.snappy#snappy-java;1.1.2.6 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   19  |   2   |   2   |   0   ||   19  |   2   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent
        confs: [default]
```

Now our command line is waiting for the data we enter the data in terminal 3:

```
C:\Users\ASUS DASH\Desktop\sem 6\bdi>kafka-console-producer.bat --broker-list localhost:9092 --topic testtopic
>hi hello welcome my name is rus hello hi once
```

And the output is obtained at terminal 4:

```
-------------------------------------------
Time:  2022-6-14 11:11:00
-------------------------------------------

-------------------------------------------
Time:  2022-6-14 11:12:00
-------------------------------------------
('is',1)
('rus',1)
('my',1)
('welcome',1)
('once',1)
('hi',2)
('hello',2)
```

CONCLUSION: Hence, with the help of the above experiment, we have integrated kafka and spark and read streaming data using Kafka.

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 10
Build sentiment analytics application using Spark Streaming AIM:

Case Study to build sentiment analytics application using Spark Streaming.

THEORY:
Build Log Analytics Application using Apache Spark Why

Apache Spark Architecture if we have Hadoop?

The Hadoop Distributed File System (HDFS), which stores files in a Hadoop-native format and parallelizes them across a cluster, and applies MapReduce the algorithm that actually processes the data in parallel. The catch here is Data Nodes are stored on disk and processing has to happen in Memory. Thus we need to do lot of I/O operations to process and also Network transfer operations happen to transfer data across the data nodes. These operations in all may be a hindrance for faster processing of data.

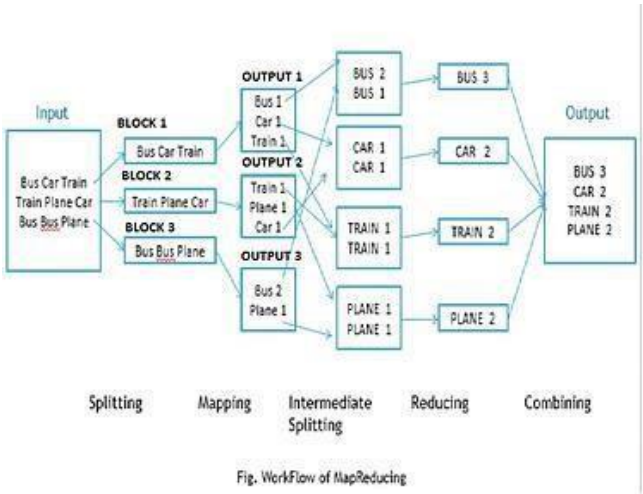
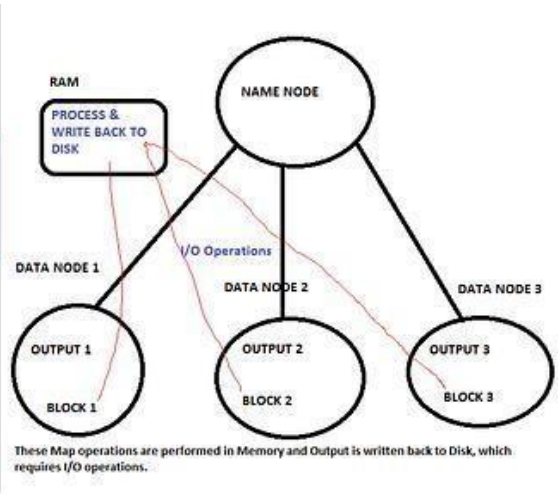


Fig. WorkFlow of MapReducing

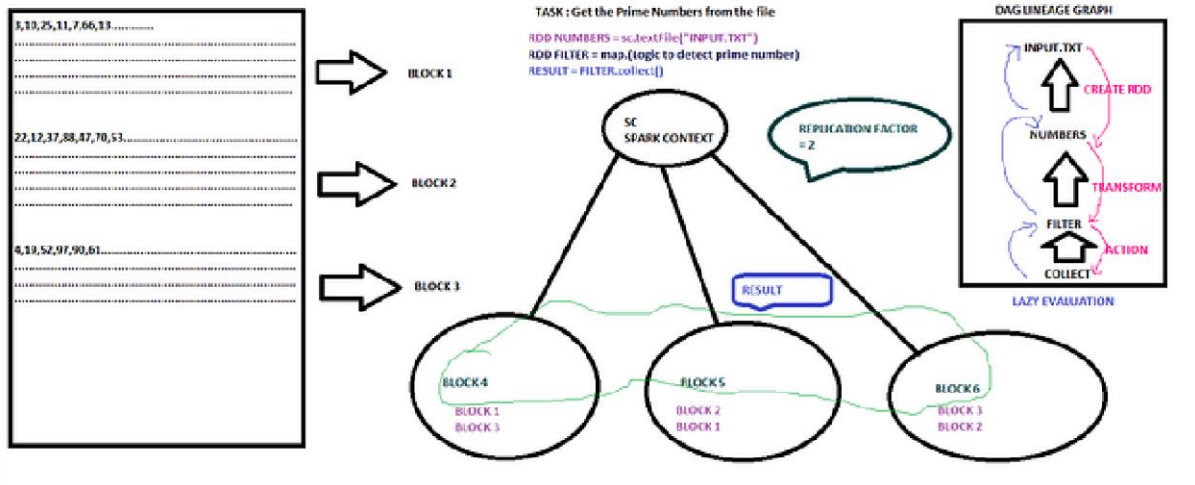


Above image describes, blocks are stored on data notes which reside on disk and for Map operation or other processing has to happen in RAM. This requires to and fro I/O Operation which causes a delay in overall result.

Apache Spark: Official website describes it as : "Apache Spark is a **fast** and **general-purpose** cluster computing system".

Fast: Apache spark is fast because computations are carried out in memory and stored there. Thus there is no picture of I/O operations as discussed in Hadoop architecture.

General-Purpose: It is an optimized engine that supports general execution graphs. It also supports a rich SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming for live data processing.



Entry point to Spark is Spark Context which handles the executors nodes. The main abstraction data structure of Spark is Resilient Distributed Dataset (RDD), which represents an immutable collection of elements that can be operated on in parallel.

Lets discuss the above example to understand better: A file consists of numbers, task is find the prime numbers from this huge chunk of numbers. If we divide them into three blocks B1,B2,B3. These blocks are immutable are stored in Memory by spark. Here the replication factor=2, thus we can see that a copy of other node is stored in corresponding other partitions. This makes it to have a fault-tolerant architecture.

Step 1 : Create RDD using Spark Context

Step 2 : Tranformation: When a map() operation is applied on these RDD, new blocks i.e B4, B5, B6 get created as new RDD's which are immutable again. This all operations happen in Memory. Note: B1,B2,B3 still exist as original.

Step 3: Action: When collect(), this when the actual results are collected and returned.

LAZY EVALUATION: Spark does not evaluate each transformation right away, but instead batch them together and evaluate all at once. At its core, it optimizes the query execution by planning out the sequence of computation and skipping potentially unnecessary steps.

Main Advantages : Increases Manageability**,** Saves Computation and increases Speed, Reduces Complexities, Optimization.

How it works ? When it we execute the code to create Spark Context, then create RDD using sc, then perform tranformation using map to create new RDD. In actual these operations are not executed in backend, rather a **Directed Acyclic Graph(DAG) Lineage** is created. Only when the **action** is performed i.e. to fetch results, example : **collect()** operation is called then it refers to DAG and climbs up to get the results, refer the figure, as climbing up it sees that filter RDD is not yet created, it climbs up to get upper results and finally reverse calculates to get the exact results.

RDD — **Resilient :** i.e. fault-tolerant with the help of RDD lineage graph. RDD's are a deterministic function of their input. This plus immutability also means the RDD's parts can be recreated at any time. This makes caching, sharing and replication easy.

Distributed : Data resides on multiple nodes.

Datasets : Represents records of the data you work with. The user can load the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.

In this experiment, we will create a Apache Access Log Analytics Application from scratch using **pyspark and SQL** functionality of Apache Spark. Python3 and latest version of pyspark.

Data Source: **ApacheAccessLog**

Prerequisite Libraries

```
pip install pyspark pip
install matplotlib pip
install numpy
```

Step 1 : As the Log Data is unstructured, we parse and create a structure from each line, which will in turn become each row while analysis.

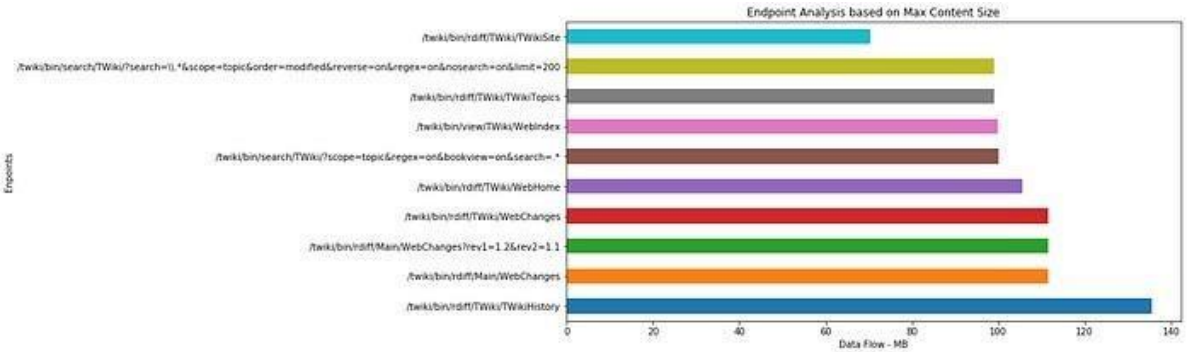
```
import re
from pyspark.sql import Row
# This is the regex which is specific to Apache Access Logs parsing, which can be modified according to
    different Log formats as per the need
# Example Apache log line:
#   127.0.0.1 - - [21/Jul/2014:9:55:27 -0800] "GET /home.html HTTP/1.1" 200 2048
#   1:IP  2:client 3:user 4:date time           5:method 6:req 7:proto   8:respcode 9:size
APACHE_ACCESS_LOG_PATTERN = '^(\S+) (\S+) (\S+) \[([\w:/]+\s[+\-]\d{4})\] "(\S+) (\S+) (\S+)" (\d{3})
    (\d+)'

# The below function is modelled specific to Apache Access Logs Model, which can be modified as per
    needs to different Logs format
# Returns a dictionary containing the parts of the Apache Access Log.
def parse_apache_log_line(logline):
    match = re.search(APACHE_ACCESS_LOG_PATTERN, logline)
    if match is None:
        raise Error("Invalid logline: %s" % logline)
    return Row(
        ip_address    = match.group(1),
        client_identd = match.group(2),
        user_id       = match.group(3),
        date = (match.group(4)[:-6]).split(":", 1)[0],
        time = (match.group(4)[:-6]).split(":", 1)[1],
        method        = match.group(5),
        endpoint      = match.group(6),
        protocol      = match.group(7),
        response_code = int(match.group(8)),
        content_size  = int(match.group(9))
    )
```

Step 2: Create Spark Context, SQL Context, DataFrame (is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database)

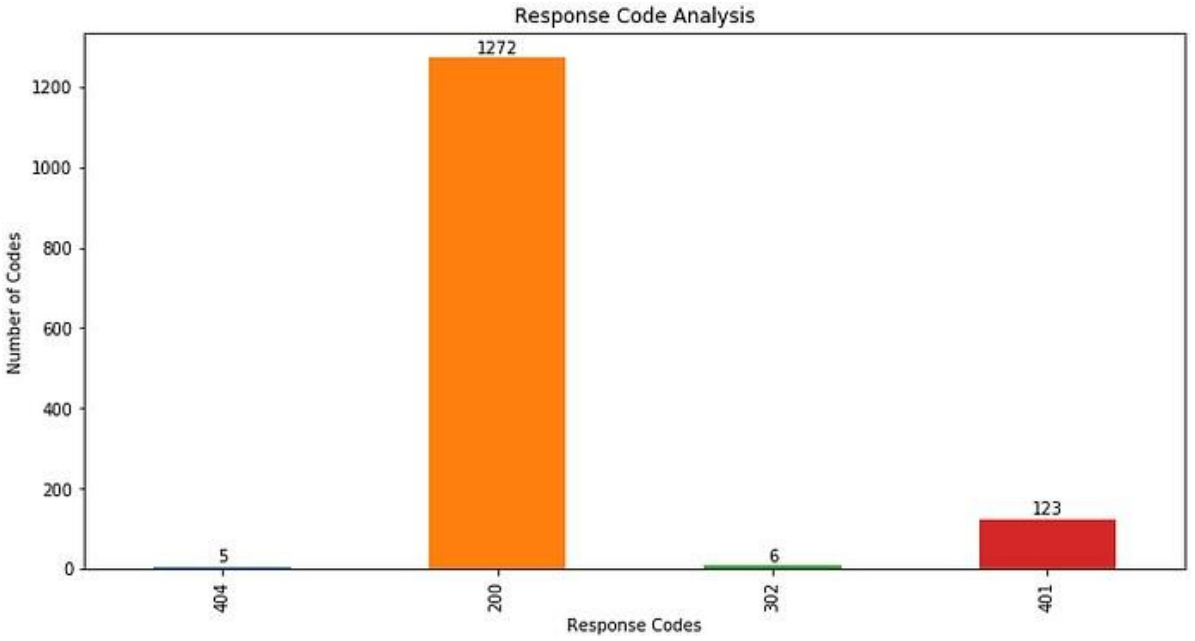
```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
import apache_access_log # This is the first file name , in which we created Data Structure of Log
import sys

# Set up The Spark App
conf = SparkConf().setAppName("Log Analyzer")
# Create Spark Context
sc = SparkContext(conf=conf)
#Create SQL Context
sqlContext = SQLContext(sc)

#Input File Path
logFile = 'Give Your Input File Path Here'

# .cache() - Persists the RDD in memory, which will be re-used again
access_logs = (sc.textFile(logFile)
               .map(apache_access_log.parse_apache_log_line)
               .cache())

schema_access_logs = sqlContext.createDataFrame(access_logs)
#Creates a table on which SQL like queries can be fired for analysis
schema_access_logs.registerTempTable("logs")
```

Step 3 : Analyze Top 10 Endpoints which Transfer Maximum Content in MB

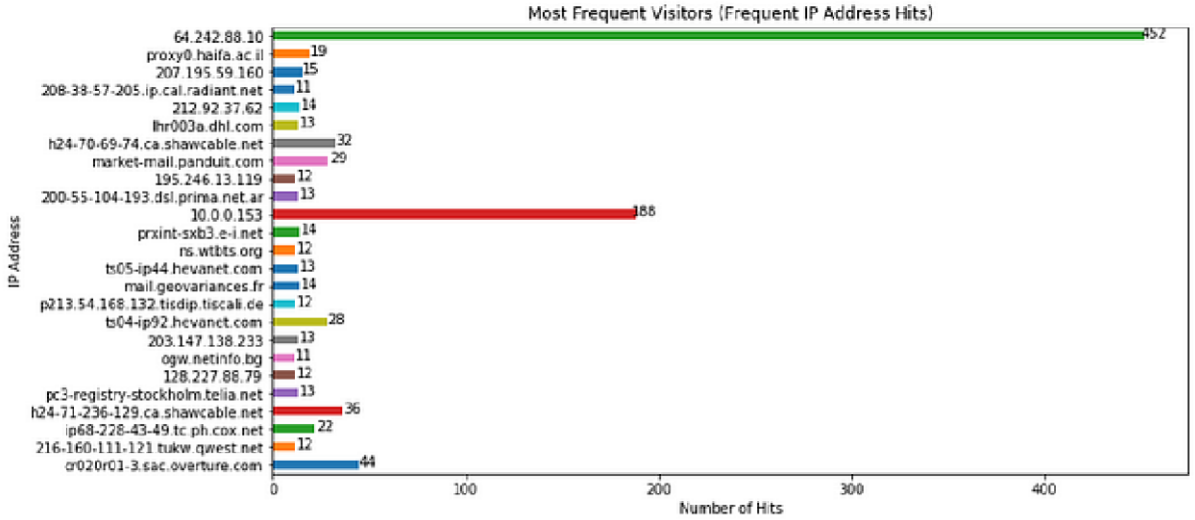
```
#Top 10 Endpoints which Transfer Maximum Content
#.rdd.map() - Will convert the resulted rows from SQL query into a map
# .collect() - actually executes the DAG to get the overall results
topEndpointsMaxSize = (sqlContext
                .sql("SELECT endpoint,content_size/1024 FROM logs ORDER BY content_size DESC LIMIT 10")
                .rdd.map(lambda row: (row[0], row[1]))
                .collect())
# Plot Analysis Code
bar_plot_list_of_tuples_horizontal(topEndpointsMaxSize,'Data Flow - MB','Enpoints','Endpoint Analysis
    based on Max Content Size')
```



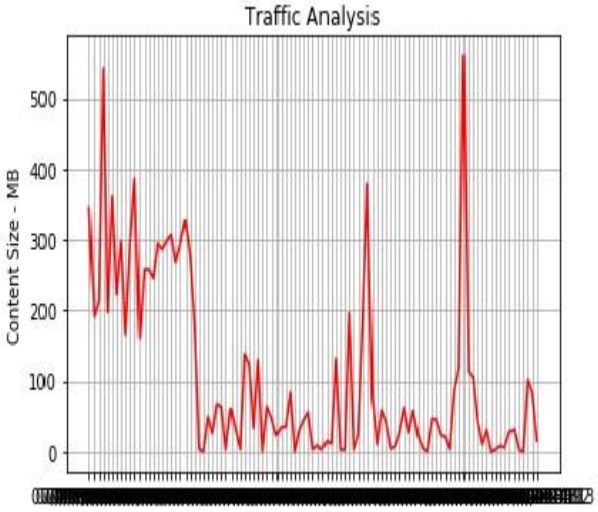
```
# Response Code Analysis
responseCodeToCount = (sqlContext
                       .sql("SELECT response_code, COUNT(*) AS theCount FROM logs GROUP BY
                           response_code")
                       .rdd.map(lambda row: (row[0], row[1]))
                       .collect())
bar_plot_list_of_tuples(responseCodeToCount,'Response Codes','Number of Codes','Response Code Analysis'
    )

# Code to Plot the results
def bar_plot_list_of_tuples(input_list,x_label,y_label,plot_title):
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    plt.figure(figsize=(12, 6))
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(plot_title)
    ax = pd.Series(y_labels).plot(kind='bar')
    ax.set_xticklabels(x_labels)
    rects = ax.patches
    for rect, label in zip(rects, y_labels):
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom')
```

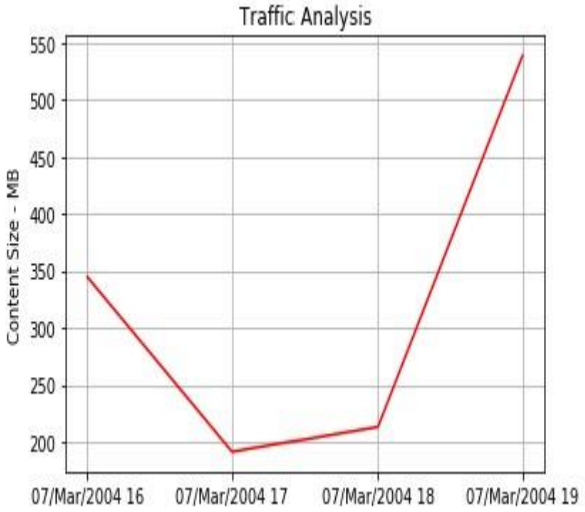



```
# Most Frequent Visitors (Most Frequent IP Address visits).
frequentIpAddressesHits = (sqlContext
                .sql("SELECT ip_address, COUNT(*) AS total FROM logs GROUP BY ip_address HAVING total >
                    10")
                .rdd.map(lambda row: (row[0], row[1]))
                .collect())
bar_plot_list_of_tuples_horizontal(frequentIpAddressesHits,'Number of Hits','IP Address','Most Frequent
    Visitors (Frequent IP Address Hits)')

```

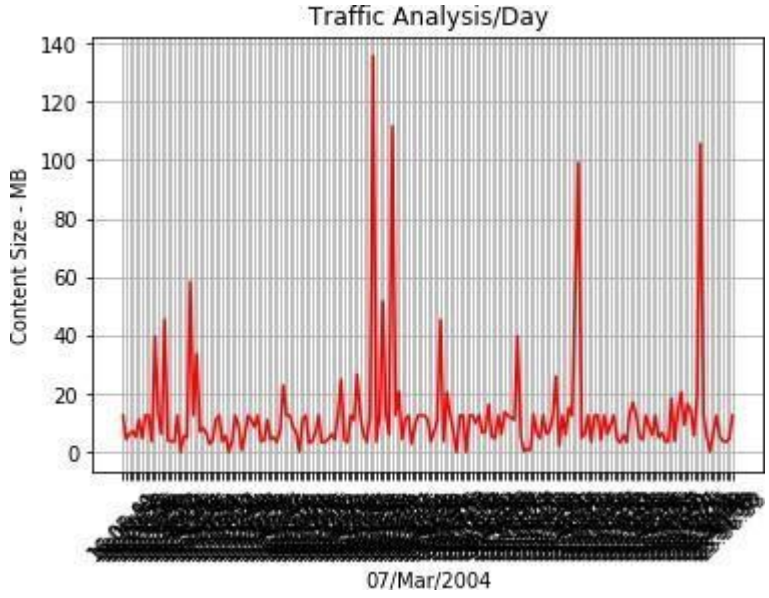



```
# Traffic Analysis for past One Week
trafficWithTime = (sqlContext
                   .sql("SELECT date, content_size/1024 FROM logs")
                   .rdd.map(lambda row: (row[0], row[1]))
                   .collect())
time_series_plot(trafficWithTime)

```

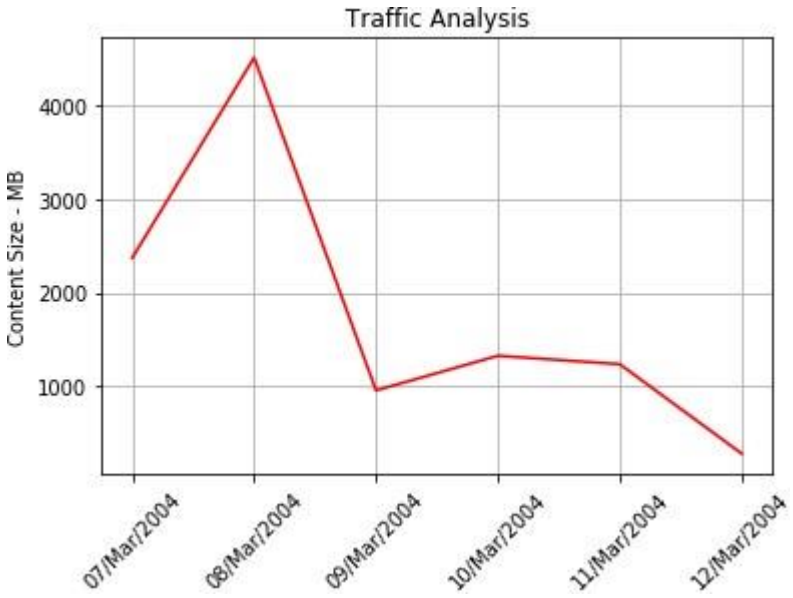





```
# Overall Traffic Analysis for a Day
Day = '07/Mar/2004'
trafficperDay = (sqlContext
                   .sql("SELECT time,content_size/1024 FROM logs where date='07/Mar/2004'")
                   .rdd.map(lambda row: (row[0], row[1]))
                   .collect())
time_series_plot(trafficperDay,Day,'Content Size - MB','Traffic Analysis/Day')
```



Outliers can be clearly detected by analysis the spikes and which end points were been hit at time by what IP Addresses.



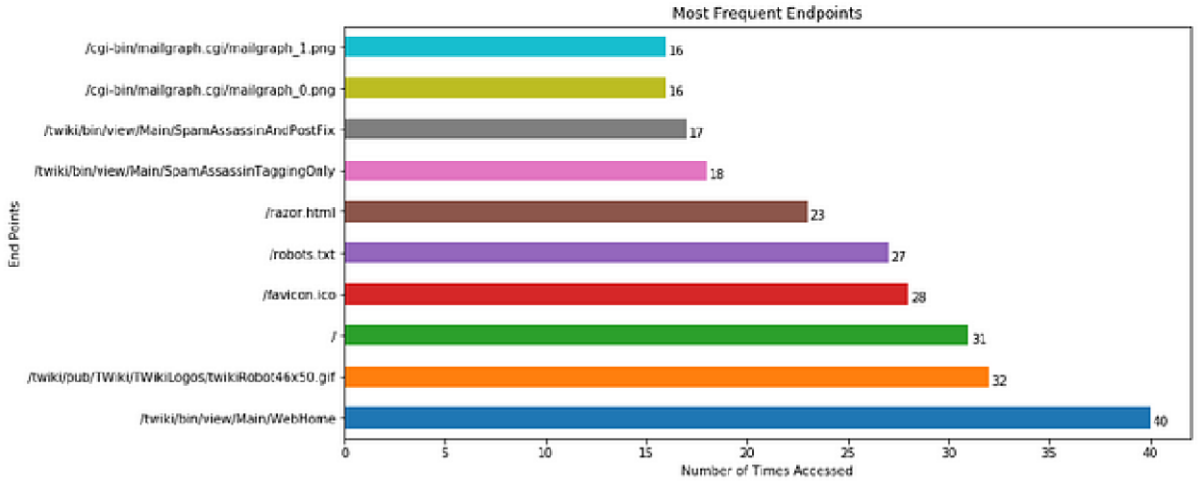
Here, we can see an unusual spike on 8th March, which can be analyzed further for identifying discrepancy.

Code for Plot Analysis:

```
def time_series_plot(input_list,x_label,y_lablel,plot_title):
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    dict_plot = OrderedDict()
    for x,y in zip(x_labels,y_labels):
       # cur_val = x.split(":", 1)[0]
        cur_val = x.split(" ")[0]
        #print(cur_val)
        dict_plot[cur_val] = dict_plot.get(cur_val, 0) + y
    input_list = list(dict_plot.items())
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    plt.plot_date(x=x_labels, y=y_labels, fmt="r-")
    plt.xticks(rotation=45)
    plt.title(plot_title)
    plt.xlabel(x_label)
    plt.ylabel(y_lablel)
    plt.grid(True)
    plt.show()
```

```

    def bar_plot_list_of_tuples_horizontal(input_list,x_label,y_label,plot_title):
    y_labels = [val[0] for val in input_list]
    x_labels = [val[1] for val in input_list]
    plt.figure(figsize=(12, 6))
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(plot_title)
    ax = pd.Series(x_labels).plot(kind='barh')
    ax.set_yticklabels(y_labels)
    for i, v in enumerate(x_labels):
        ax.text(int(v) + 0.5, i - 0.25, str(v),ha='center', va='bottom')
```

```

   # Frequent End Points
topEndpoints = (sqlContext
                .sql("SELECT endpoint, COUNT(*) AS total FROM logs GROUP BY endpoint ORDER BY total
                    DESC LIMIT 10")
                .rdd.map(lambda row: (row[0], row[1]))
                .collect())
bar_plot_list_of_tuples_horizontal(topEndpoints,'Number of Times Accessed','End Points','Most
    Frequent Endpoints')
```



CONCLUSION: Hence, with the help of the above experiment, we successfully built a log analytics application as a case study for Spark streaming. We visualized most frequent visitors, traffic analysis for a day and past one week through our implemented application.

Experiment No:

SAP ID: 60004200107

Q1. Consider any 1 case study of big data solution. State its characteristics in terms of 10 Vs of Bi

JP Morgan can use big data in several ways. The objectives are

- optimizing the sales of foreclosed properties
- Develop new marketing initiatives
- Credit assessment.
- Identify possible opportunities to make money

(i) Volume

Hadoop is designed to handle large volumes of data and JPMC uses Hadoop to store and process vast amounts of financial data, including market data and customer data.

(ii) Velocity

Hadoop has several tools such as Apache Storm Spark that can perform real time data processing and analytics

(iii) Variety

Hadoop is a flexible platform that can handle various types of data. It is utilised by JPMC to store and interpret data from several sources

(iv) Veracity

Financial data used by JPMC must be precise and can use Apache Nifi and Apache Atlas

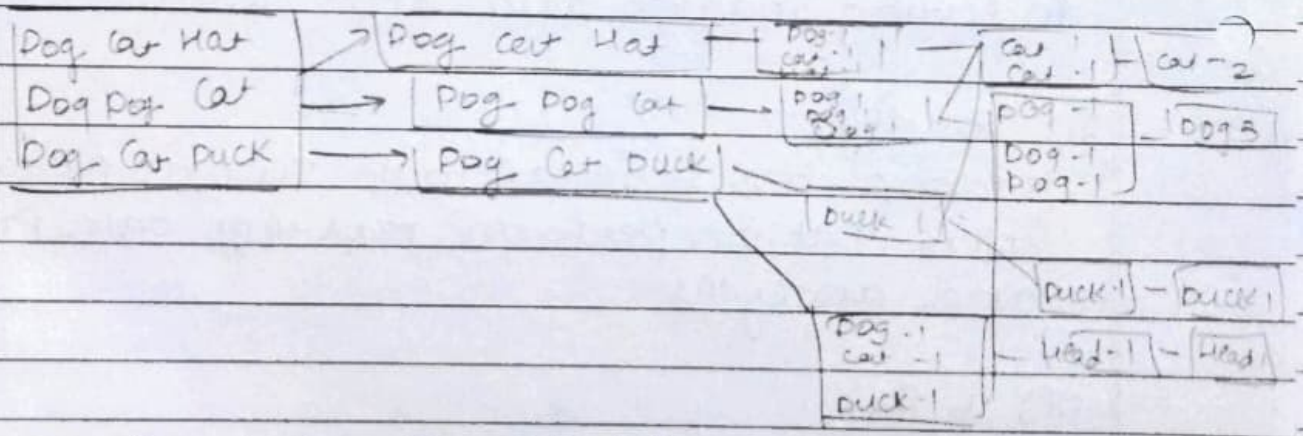
(v) Value

JPMC may get insight into consumer behavior, market trends and potential hazards by keeping and processing vast volumes of ~~data~~ data.

Q2 Explain concept of MapReduce to find frequent words

MapReduce is maploop framework and programming model for processing big data using automatic parallelization & distribution.

The output of map tasks is used as input in reduce tasks and the data is shuffled and reduced.



The input data is divided into multiple segments, the processed. In parallel to reduce processing. Data will be divided into three input splits so that

work can be distributed over map tasks
- The mapper counts the number of times each word occurs from input split the form of key value pairs where the key is the word and value is frequency.
- The shuffler phase in which the values are grouped by keys in the form of key value pairs.
- Next reducer is used for pairs with same key.
- Final output is displayed as frequency of words.

Q3 Advance Indexing in HBASE

- In HBASE the row key provides the same data retrieval benefits as a primary index. So when you create a secondary index, use elements that are different from row key.
- Secondary indexes allow you to have a secondary way to read an HBASE table. They provide a way to efficiently access records by means of some piece of information other than primary key.
- They require additional cluster space and processing because the act of creating a secondary index requires both space and processing.
- A method of index maintenance called diff index can help. Eg. SQL create secondary index for Hbase, maintain those indexes are speed up queries.

Assignment - 2

60004200107

Q1. Write program for RDD to find cube of number and display greater than 10.

```
from pyspark import SparkContext, SparkConf
conf = SparkConf().setAppName("cube RDD").
        setMaster("local")
sc = SparkContext(conf=conf)
numbers = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
cubes = numbers.map(lambda x: x**3)
greaterthan10 = cubes.filter(lambda x: x>10)
print(greaterthan10.collect())
```

Q2. Industry use cases of NoSQL

1) E-commerce : widely used because of large volumes of unstructured data, user profiles, transaction history, product catalogues.

2) Social media to store user generated content, included posts, comments, likes, real time analytics.

3) Gaming : online gaming to handle massive data amounts, game logs, and real time gameplay data.

4. Healthcare: Store patient data, medical records, diagnostic test results, and medical images.

5. Internet of things: Handle large data generated by connected devices, sensor data, device states.

Q3 Industry use cases of Apache Kafka & Apache Storm

1. Financial services: real time processing of financial data, eg stock prices, trading volumes, transaction data, analyze trends.

2. Telecommunication: Handle large data by mobile devices, call records, text messages, location data, analyze network traffic

3. Retail: real time processing of customer data, purchase history, browser behavior, recommendations. Analyze customer behaviour.

4. Healthcare: real time processing of patient data, medical records, medical images. Analyze patient data in real time

5. Manufacturing: Use for real time monitoring of equipment performance and predictive maintenance.