CCS334 BDA Lab Manual

Big Data Analytics (Anna University)

MISRIMAL NAVAJEE MUNOTH JAIN ENGINEERING COLLEGE

OWNED AND MANAGED BY TAMILNADU EDUCATIONAL AND MEDICAL FOUNDATION

A Jain Minority Institution

Approved by AICTE &Programmes Accredited by NBA, New Delhi, (UG Programmes – MECH, AI&DS, ECE, CSE,IT)

All Programmes Recognized by the Government of Tamil Nadu and Affiliated to Anna University, Chennai

Guru MarudharKesari Building, Jyothi Nagar, Rajiv Gandhi Salai, OMR Thoraipakkam, Chennai – 600 097.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CCS334 BIG DATA ANALYTICS LABORATORY

REGULATION-2021

NAME :

REGISTER NUMBER :

YEAR/SEMESTER : III/V

MISRIMAL NAVAJEE MUNOTH JAIN ENGINEERING COLLEGE

OWNED AND MANAGED BY TAMILNADU EDUCATIONAL AND MEDICAL FOUNDATION

A Jain Minority Institution

Approved by AICTE &Programmes Accredited by NBA, New Delhi, (UG Programmes – MECH, AI&DS, ECE, CSE,IT)

All Programmes Recognized by the Government of Tamil Nadu and Affiliated to Anna University, Chennai

Guru MarudharKesari Building, Jyothi Nagar, Rajiv Gandhi Salai, OMR Thoraipakkam, Chennai – 600 097.

.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

VISION

To produce high quality, creative and ethical engineers, and technologists contributing effectively to the ever-advancing Artificial Intelligence and Data Science field.

MISSION

To educate future software engineers with strong fundamentals by continuously improving the teaching-learning methodologies using contemporary aids.

To produce ethical engineers/researchers by instilling the values of humility, humaneness, honesty and courage to serve the society.

To create a knowledge hub of Artificial Intelligence and Data Science with everlasting urge to learn by developing, maintaining and continuously improving the resources/Data Science.

MISRIMAL NAVAJEE MUNOTH JAIN ENGINEERING COLLEGE

OWNED AND MANAGED BY TAMILNADU EDUCATIONAL AND MEDICAL FOUNDATION

A Jain Minority Institution

Approved by AICTE &Programmes Accredited by NBA, New Delhi, (UG Programmes – MECH, AI&DS, ECE, CSE,IT)

All Programmes Recognized by the Government of Tamil Nadu and Affiliated to Anna University, Chennai

Guru MarudharKesari Building, Jyothi Nagar, Rajiv Gandhi Salai, OMR Thoraipakkam, Chennai – 600 097.

Register No:												

BONAFIDE CERTIFICATE

This is to certify that this is a bonafide record of the work done by Mr./Ms._____ of III YEAR/ V SEM B.Tech- **ARTIFICIAL INTELLIGENCE AND DATA SCIENCE** in **CCS334- BIG DATA ANALYTICS LABORATORY** during the Academic year 2023 – 2024.

Faculty-in-charge **Head of the Department**

Submitted for the University Practical Examination held on :__/__/____

Internal Examiner **External Examiner**

DATE: **DATE:**

MISRIMAL NAVAJEE MUNOTH JAIN ENGINEERING COLLEGE

OWNED AND MANAGED BY TAMILNADU EDUCATIONAL AND MEDICAL FOUNDATION

A Jain Minority Institution

Approved by AICTE &Programmes Accredited by NBA, New Delhi, (UG Programmes – MECH, EEE, ECE, CSE & IT)

All Programmes Recognized by the Government of Tamil Nadu and Affiliated to Anna University, Chennai

Guru MarudharKesari Building, Jyothi Nagar, Rajiv Gandhi Salai, OMR Thoraipakkam, Chennai – 600 097.

CCS334 BIG DATA ANALYTICS LABORATORY

COURSE OUTCOMES

CO1	Describe big data and use cases from selected business domains.
CO 2	Explain NoSQL big data management.
CO 3	Install, configure, and run Hadoop and HDFS.
CO 4	Perform map-reduce analytics using Hadoop.
CO 5	Use Hadoop-related tools such as HBase, Cassandra, Pig, and Hive for big data analytics

CCS334 BIG DATA ANALYTICS LABORATORY

CONTENT

S.NO.	EXPERIMENTS	PAGE NO	DATE	SIGNATURE
1.	Downloading and installing Hadoop; Understanding different Hadoop modes. Startup scripts, Configuration files.			
2.	Hadoop Implementation of file management tasks, such as Adding files and directories, retrieving files and Deleting files			
3.	Implement of Matrix Multiplication with Hadoop Map Reduce			
4.	Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.			
5.	Installation of Hive along with practice examples.			
6.	Installation of HBase along with Practice examples.			
7.	Installing thrift.			
8.	Practice importing and exporting data from various databases.			

Syllabus

CCS334 BIG DATA ANALYTICS LABORATORY

COURSE OBJECTIVES:

To understand big data.
To learn and use NoSQL big data management.
To learn mapreduce analytics using Hadoop and related tools.
To work with map reduce applications
To understand the usage of Hadoop related tools for Big Data Analytics

Tools: Cassandra, Hadoop, Java, Pig, Hive and HBase.

Suggested Exercises:

1.Downloading and installing Hadoop; Understanding different Hadoop modes. Startup scripts, Configuration files.
2. Hadoop Implementation of file management tasks, such as Adding files and directories, retrieving files and Deleting files
3. Implement of Matrix Multiplication with Hadoop Map Reduce
4. Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.
5. Installation of Hive along with practice examples.
6. Installation of HBase, Installing thrift along with Practice examples
7. Practice importing and exporting data from various databases.

EXP.NO:1 DATE:	DOWNLOADING AND INSTALLING HADOOP; UNDERSTANDING DIFFERENT HADOOP MODES. STARTUP SCRIPTS, CONFIGURATION FILES.

AIM:

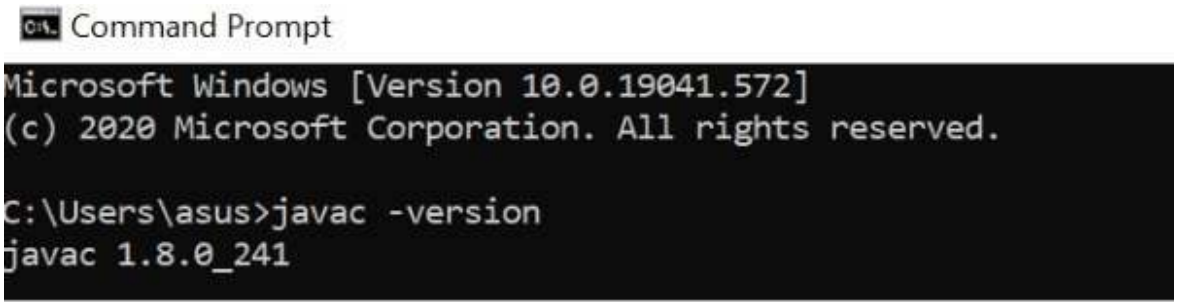
To Downloading and installing Hadoop; Understanding different Hadoop modes. Startup scripts, Configuration files.

PREREQUISITES TO INSTALL HADOOP ON WINDOWS

- **VIRTUAL BOX** (For Linux): it is used for installing the operating system on it.
- **OPERATING SYSTEM**: You can install Hadoop on Windows or Linux based operating systems. Ubuntu and CentOS are very commonly used.
- **JAVA**: You need to install the Java 8 package on your system.
- **HADOOP**: You require Hadoop latest version

1. **Install Java**

- Java JDK Link to download https://www.oracle.com/java/technologies/javase-jdk8-downloads.html
- Extract and install Java in C:\Java
- Open cmd and type -> javac –version

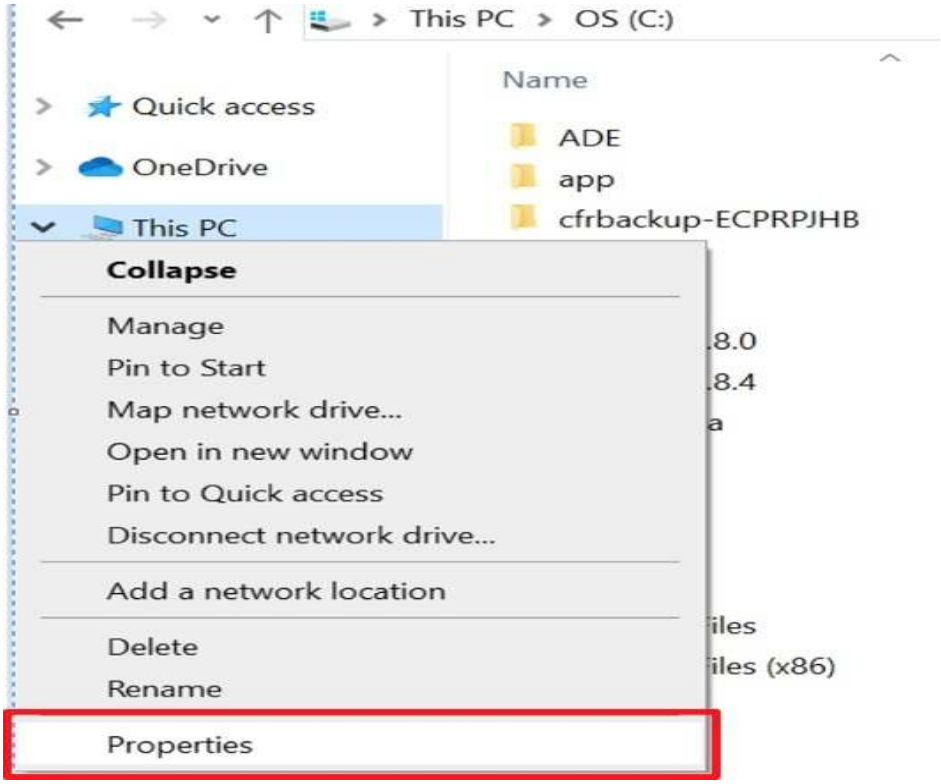


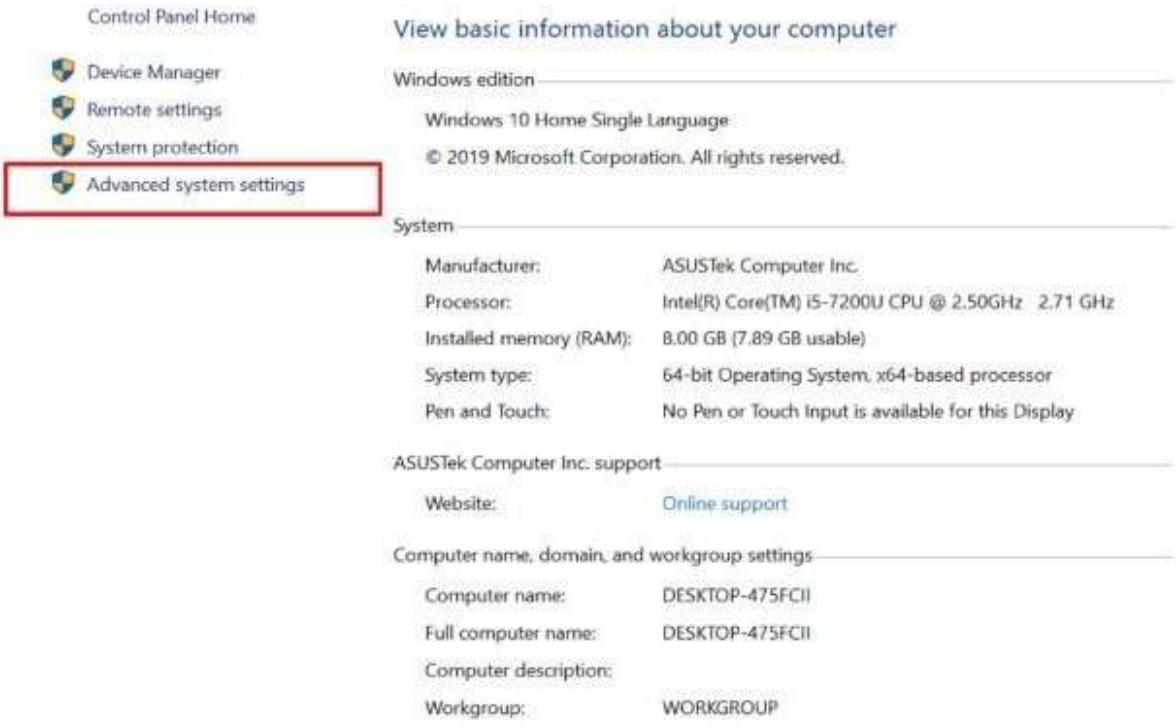
2. **Download Hadoop**

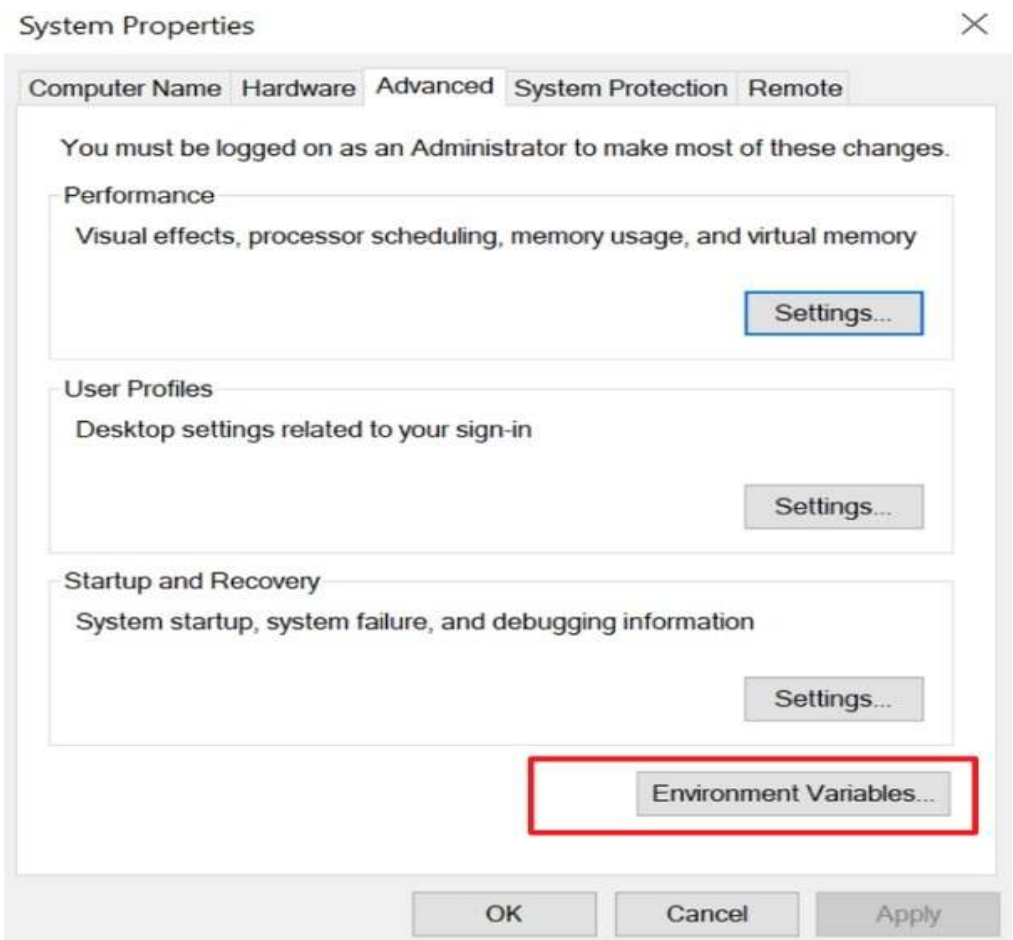
https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz

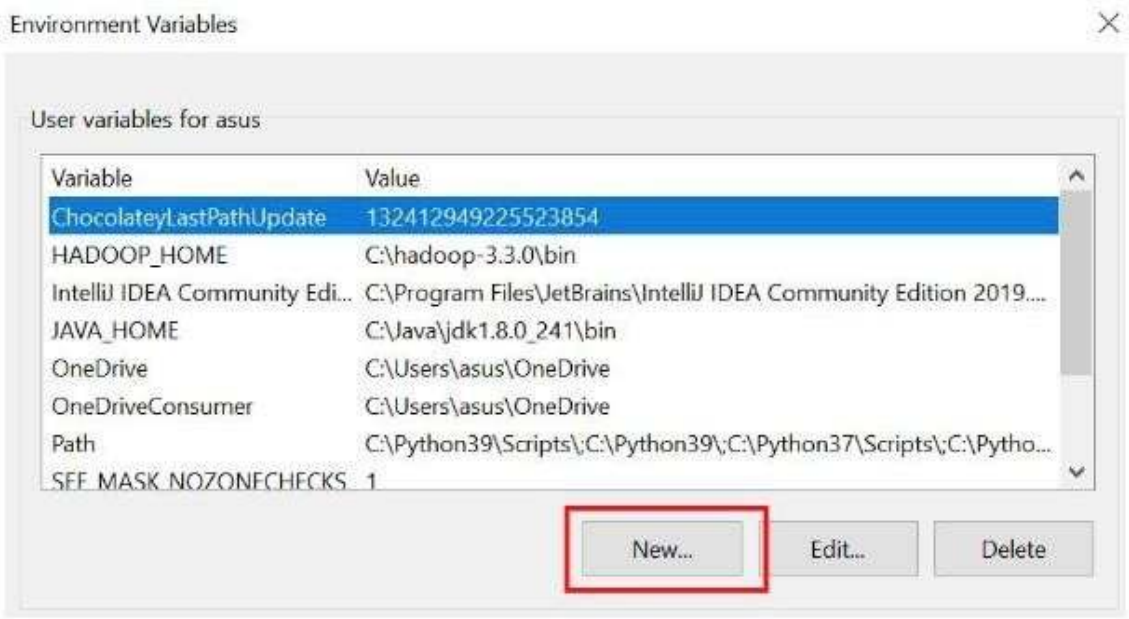
- extract to C:\Hadoop

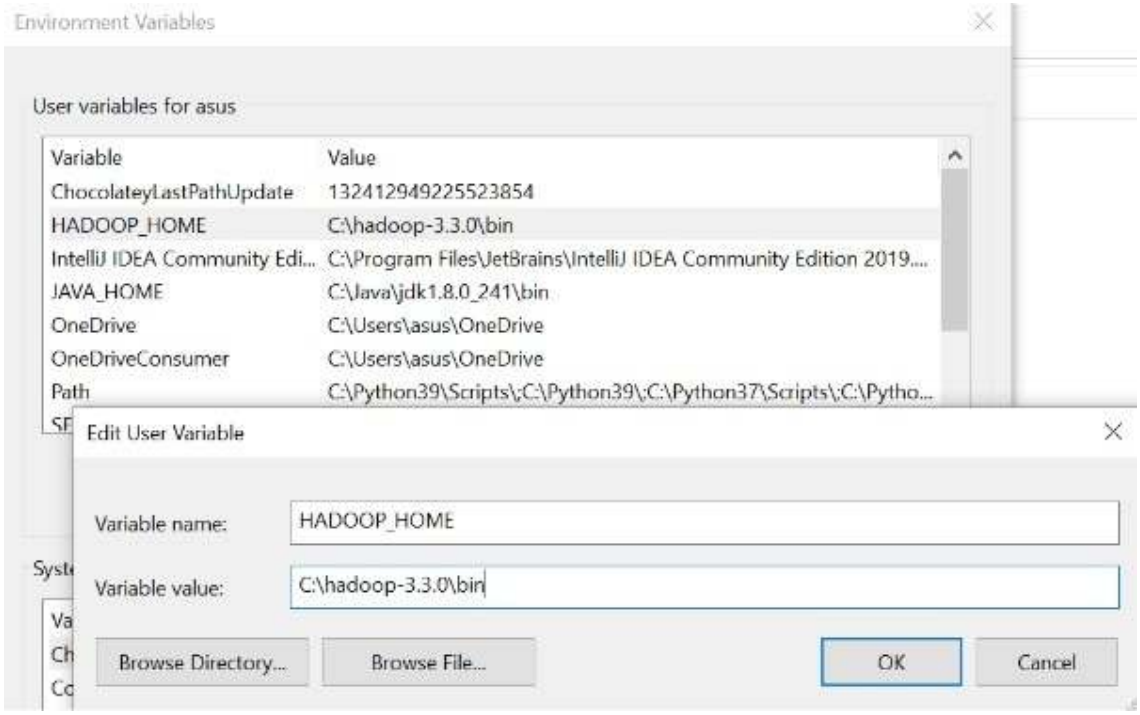
3. Set the path JAVA_HOME Environment variable
4. Set the path HADOOP_HOME Environment variable

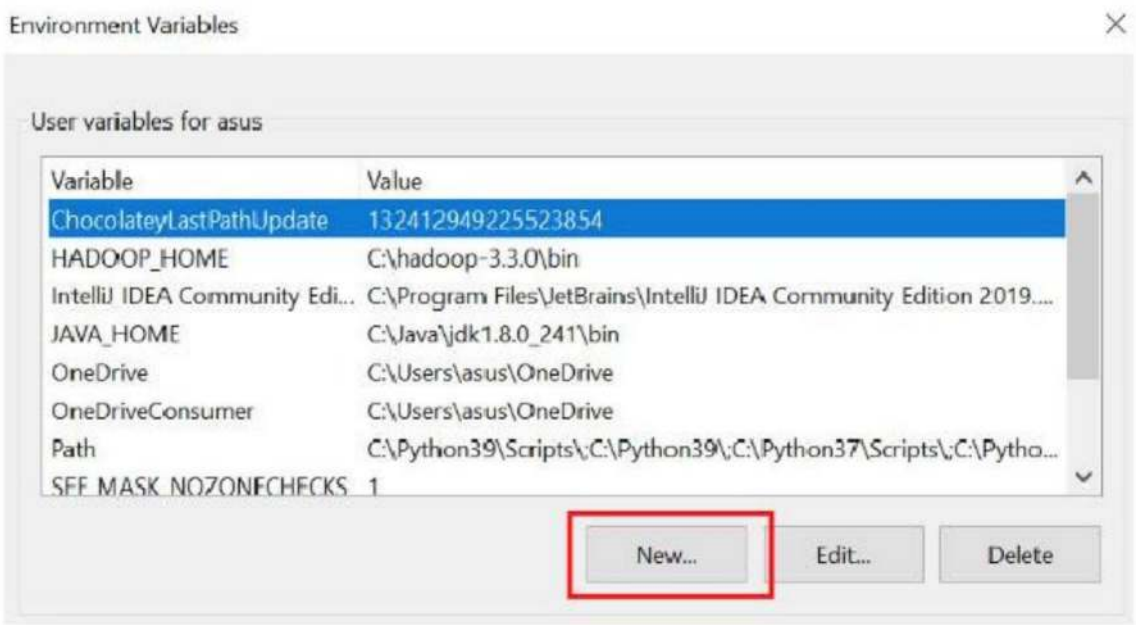


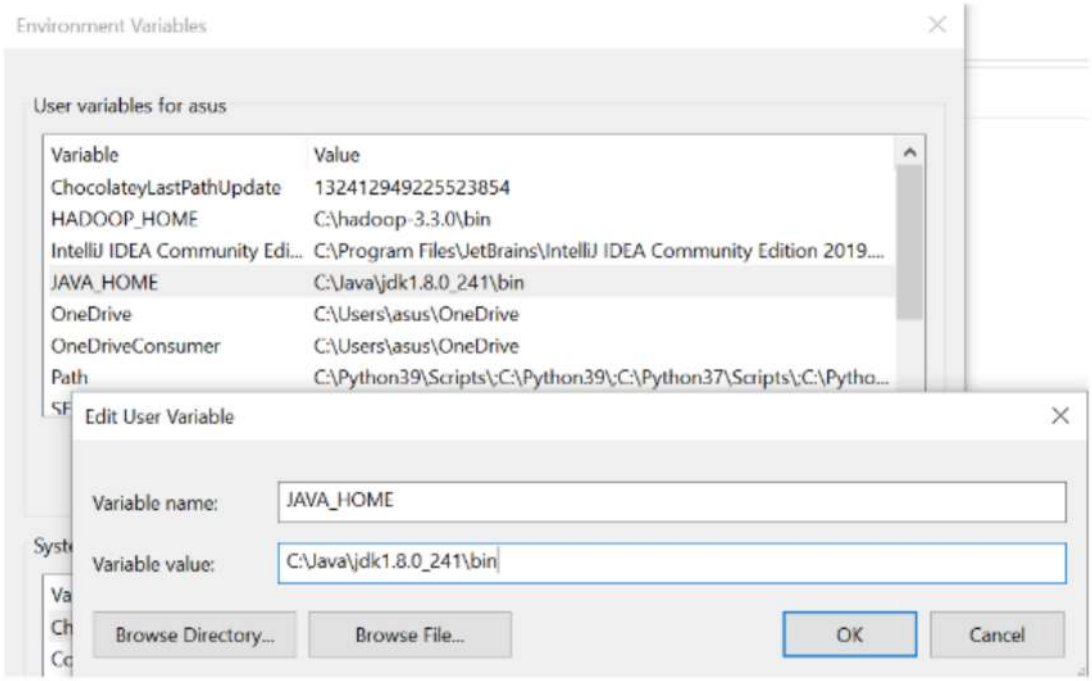


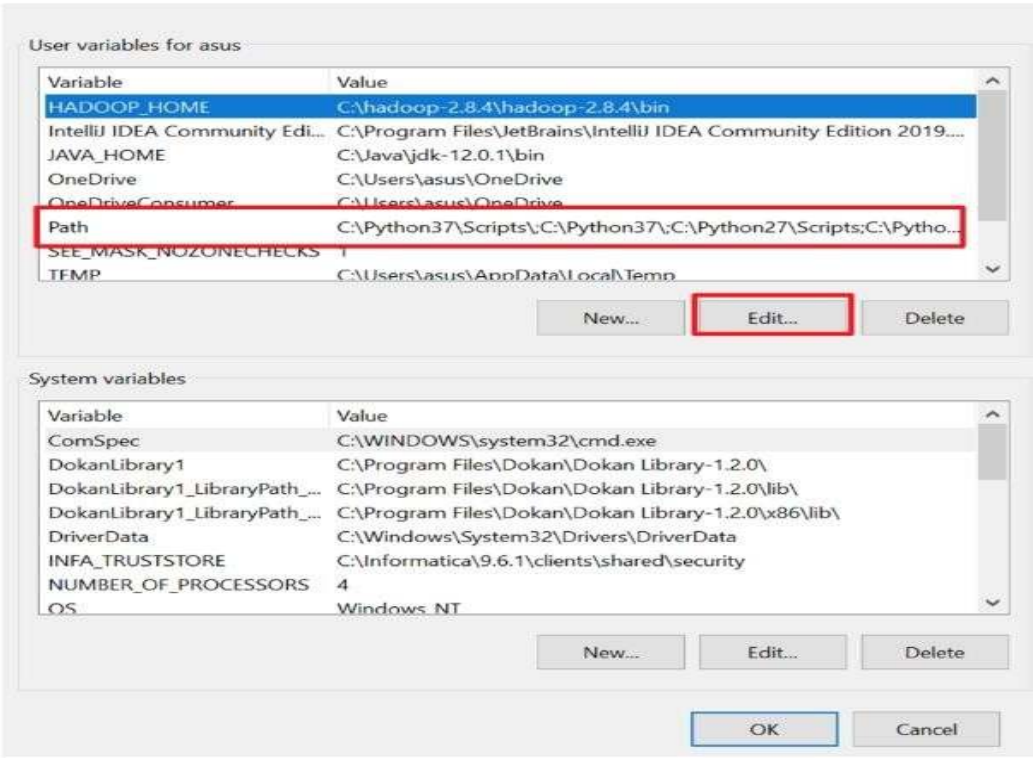


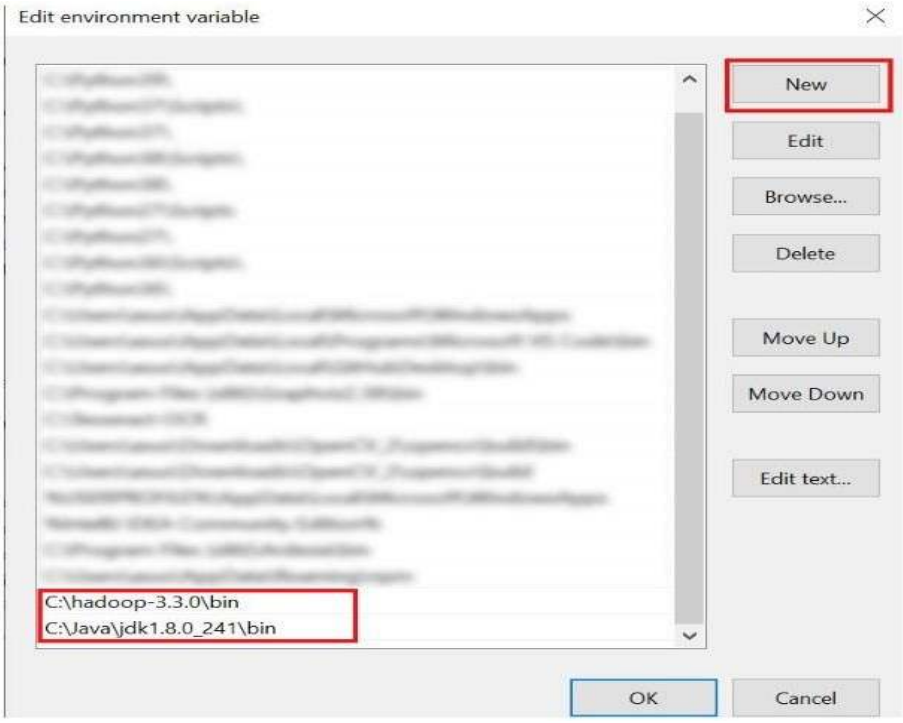












5. Configurations

Edit file C:/Hadoop-3.3.0/etc/hadoop/core-site.xml,

paste the xml code in folder and save

```
<configuration>
  <property>
     <name>fs.defaultFS</name>
     <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

==========

Rename “mapred-site.xml.template” to “mapred-site.xml” and edit this file C:/Hadoop-3.3.0/etc/hadoop/mapred-site.xml, paste xml code and save this file.

```
<configuration>
  <property>
     <name>mapreduce.framework.name</name>
     <value>yarn</value>
  </property>
</configuration>
```

==========

Create folder “data” under “C:\Hadoop-3.3.0”

Create folder “datanode” under “C:\Hadoop-3.3.0\data”

Create folder “namenode” under “C:\Hadoop-3.3.0\data”

==========

Edit file C:\Hadoop-3.3.0/etc/hadoop/hdfs-site.xml,

paste xml code and save this file.

```
<configuration>
  <property>
     <name>dfs.replication</name>
     <value>1</value>
  </property>
  <property>
     <name>dfs.namenode.name.dir</name>
     <value>/hadoop-3.3.0/data/namenode</value>
  </property>
  <property>
     <name>dfs.datanode.data.dir</name>
```

```
    <value>/hadoop-3.3.0/data/datanode</value>
  </property>
</configuration>
```

==========

Edit file C:/Hadoop-3.3.0/etc/hadoop/yarn-site.xml,

paste xml code and save this file.

```
<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>
</configuration>
```

==========

Edit file C:/Hadoop-3.3.0/etc/hadoop/hadoop-env.cmd

by closing the command line

"JAVA_HOME=%JAVA_HOME%" instead of set "JAVA_HOME=C:\Java"

---

6. Hadoop Configurations

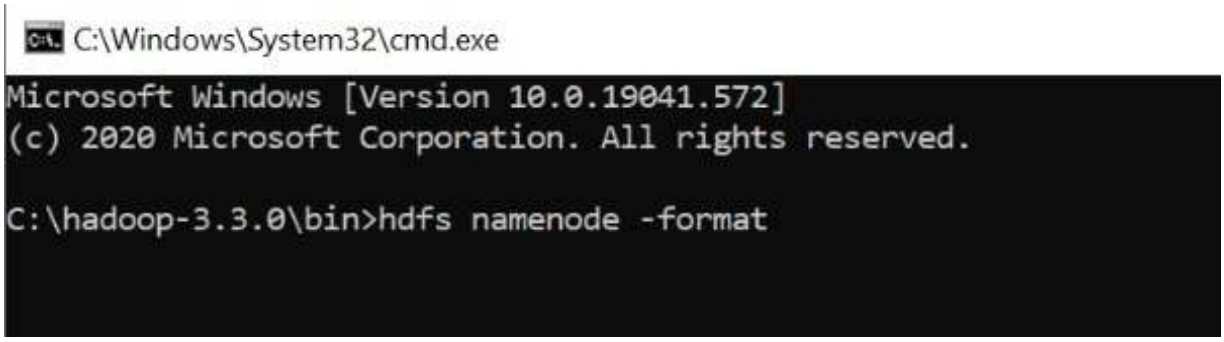
Download

https://github.com/brainmentorspvtltd/BigData_RDE/blob/master/Hadoop%20Configuration.zip

or (for hadoop 3)

https://github.com/s911415/apache-hadoop-3.1.0-winutils

- Copy folder bin and replace existing bin folder in C:\Hadoop-3.3.0\bin
- Format the NameNode
- Open cmd and type command "hdfs namenode –format"



7. **Testing**

- Open cmd and change directory to C:\Hadoop-3.3.0\sbin
- type start-all.cmd

(Or you can start like this)

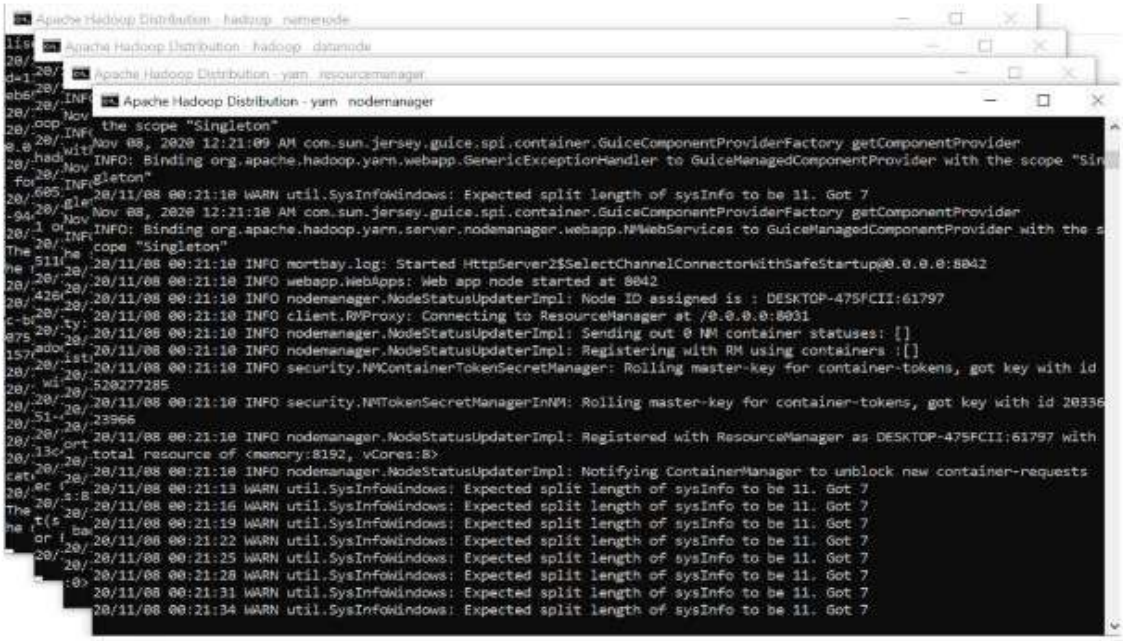
Start namenode and datanode with this command

- type start-dfs.cmd
- Start yarn through this command
- type start-yarn.cmd

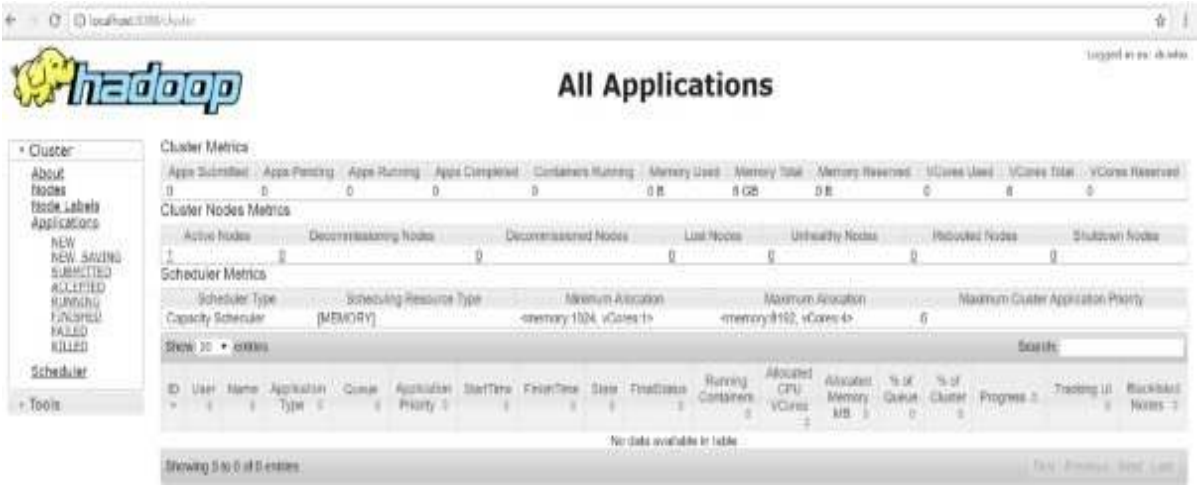
Make sure these apps are running

- Hadoop Namenode
- Hadoop datanode
- YARN Resource Manager

– YARN Node Manager



Open: http://localhost:8088



Open: http://localhost:9870

RESULT:

Downloaded and installed Hadoop and also understand different Hadoop modes. Startup scripts, Configuration files are successfully implemented.

EXP.NO:2	HADOOP IMPLEMENTATION OF FILE MANAGEMENT TASKS, SUCH AS ADDING FILES AND DIRECTORIES, RETRIEVING FILES AND DELETING FILES.
DATE:	

AIM:

To implement the following file management tasks in Hadoop:

1. Adding files and directories
2. Retrieving files
3. Deleting Files

1.Create a directory in HDFS at given path(s).

Usage:

hadoop fs -mkdir <paths> Example:

hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2

2.List the contents of adirectory.

Usage :

hadoop fs -ls <args>

Example:

hadoop fs -ls /user/saurzcode

3.Upload and download a file in HDFS.

Upload: hadoop fs -put:

Copy single src file, or multiple src files from local file system to the Hadoop data file system

Usage:

hadoop fs -put <localsrc> ... <HDFS_dest_Path> Example:

hadoop fs -put /home/saurzcode/Samplefile.txt /user/ saurzcode/dir3/

Download:

hadoop fs -get:

Copies/Downloads files to the local file system

Usage:

hadoop fs -get <hdfs_src> <localdst> Example:

hadoop fs -get /user/saurzcode/dir3/Samplefile.txt /home/

4.See contents of a file Same

as unix cat command: Usage:

hadoop fs -cat <path[filename]>

Example:

hadoop fs -cat /user/saurzcode/dir1/abc.txt

1. **Copy a file from source todestination**

This command allows multiple sources as well in which casethe destination must be a directory.

Usage:

hadoop fs -cp <source> <dest>

Example:

hadoop fs -cp /user/saurzcode/dir1/abc.txt /user/saurzcode/dir2

2. **Copy a file from/To Local file system to HDFS**

copyFromLocal

Usage:

hadoop fs -copyFromLocal <localsrc>

URI Example:

hadoop fs -copyFromLocal /home/saurzcode/abc.txt /user/ saurzcode/abc.txt

Similar to put command, except that the source is restricted to a local file reference.

copyToLocal

Usage:

hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

Similar to get command, except that the destination is restricted to a local file reference.

3. **Move file from source to destination.**

Note:- Moving files across filesystem is not permitted.

Usage :

hadoop fs -mv <src> <dest> Example:

hadoop fs -mv /user/saurzcode/dir1/abc.txt /user/saurzcode/ dir2

4. **Remove a file or directory in HDFS.**

Remove files specified as argument. Deletes directory only when it is empty

Usage :

hadoop fs -rm <arg> Example:

hadoop fs -rm /user/saurzcode/dir1/abc.txt

Recursive version of delete.

Usage :

hadoop fs -rmr <arg> Example:

hadoop fs -rmr /user/saurzcode/

5. **Display last few lines of a file.**

Similar to tail command in Unix.

Usage :

hadoop fs -tail <path[filename]> Example:

hadoop fs -tail /user/saurzcode/dir1/abc.txt

6. **Display the aggregate length of a file.**

Usage :

hadoop fs -du <path> Example:

hadoop fs -du /user/saurzcode/dir1/abc.txt

RESULT:

Thus,the Hadoop Implementation of file management tasks, such as Adding files and directories, retrieving files and Deleting files is executed successfully.

EXP.NO:3 DATE:	IMPLEMENT OF MATRIX MULTIPLICATION WITH HADOOP MAP REDUCE

AIM:

To write a Map Reduce Program that implements Matrix Multiplication.

ALGORITHM:

We assume that the input matrices are already stored in Hadoop Distributed File System (HDFS) in a suitable format (e.g., CSV, TSV) where each row represents a matrix element. The matrices are compatible for multiplication (the number of columns in the first matrix is equal to the number of rows in the second matrix).

STEP 1: MAPPER

The mapper will take the input matrices and emit key-value pairs for each element in the result matrix. The key will be the (row, column) index of the result element, and the value will be the corresponding element value.

STEP 2: REDUCER

The reducer will take the key-value pairs emitted by the mapper and calculate the partial sum for each element in the result matrix.

STEP 3: MAIN DRIVER

The main driver class sets up the Hadoop job configuration and specifies the input and output paths for the matrices.

STEP 4: RUNNING THE JOB

To run the MapReduce job, you need to package your classes into a JAR file and then submit it to Hadoop using the hadoop jar command. Make sure to replace input_path and output_path with the actual HDFS paths to your input matrices and desired output directory.

PROGRAM:

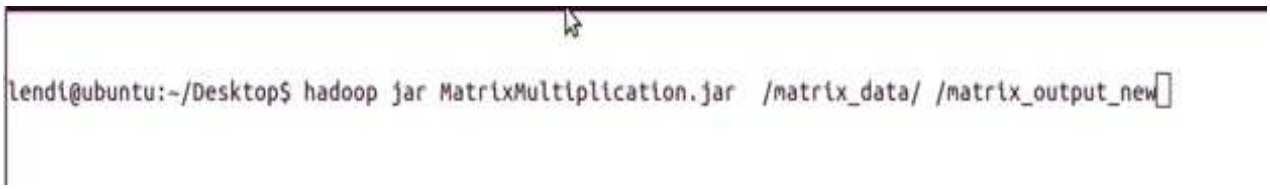
```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

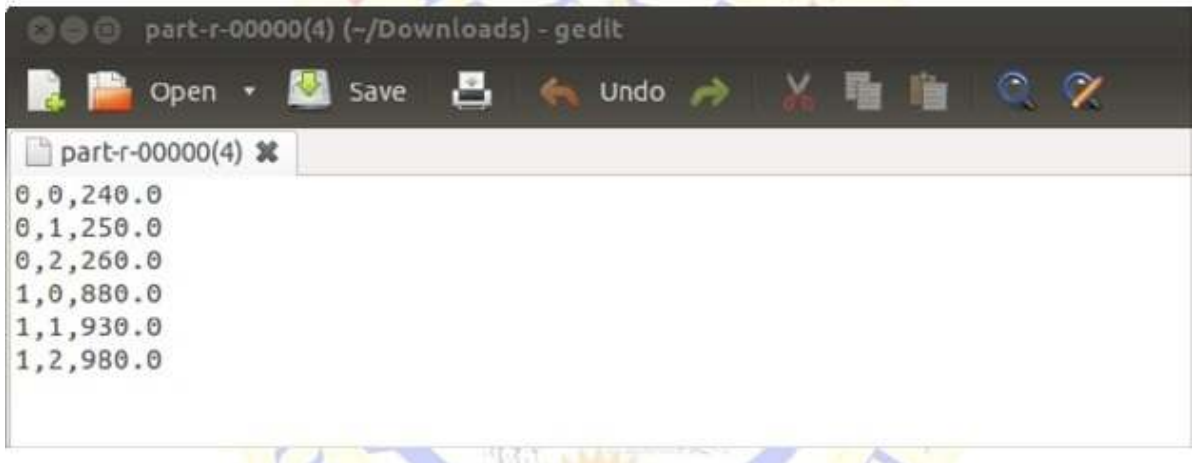
```
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;
public class MatrixMultiplicationMapper extends Mapper<LongWritable, Text, Text, Text>
{
protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        // Parse the input line to get row, column, and value of each element in the input matrices
        String[] elements = value.toString().split(",");
        int row = Integer.parseInt(elements[0]);
        int col = Integer.parseInt(elements[1]);
        int val = Integer.parseInt(elements[2]);
        // Emit key-value pairs where key is (row, column) index of the result element
        // and value is the corresponding element value
        context.write(new Text(row + "," + col), new Text(val));
    }
}
public class MatrixMultiplicationReducer extends Reducer<Text, Text, Text, IntWritable>
{ protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
        int result = 0;
        for (Text value : values) {
            // Accumulate the partial sum for the result element
            result += Integer.parseInt(value.toString());
        }
        // Emit the final result for the result element
        context.write(key, new IntWritable(result));
    }
```

```
}
public class MatrixMultiplicationDriver {
    public static void main(String[] args) throws Exception
        { Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Matrix Multiplication");
        job.setJarByClass(MatrixMultiplicationDriver.class);
        job.setMapperClass(MatrixMultiplicationMapper.class);
        job.setReducerClass(MatrixMultiplicationReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Run the program

hadoop jar matrixmultiplication.jar MatrixMultiplicationDriver input_path output_path





RESULT:

Thus the Map Reduce Program that implements Matrix Multiplication was executed and verified successfully.

EXP.NO:4	RUN A BASIC WORD COUNT MAP REDUCE PROGRAM TO UNDERSTAND MAP REDUCE PARADIGM
DATE:	

AIM:

To write a Basic Word Count program to understand Map Reduce Paradigm.

ALGORITHM:

The entire MapReduce program can be fundamentally divided into three parts:

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

STEP 1: MAPPER CODE:

We have created a class Map that extends the class Mapper which is already defined in the MapReduce Framework.

- We define the data types of input and output key/value pair after the class declaration using angle brackets.
- Both the input and output of the Mapper is a key/value pair.

Input:

- The key is nothing but the offset of each line in the text file:LongWritable
- The value is each individual line : Text

Output:

- The key is the tokenized words: Text
- We have the hardcoded value in our case which is 1: IntWritable
- **Example –** Dear 1, Bear 1, etc.

We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to 1.

STEP 2 : REDUCER CODE:

- We have created a class Reduce which extends class Reducer like that of Mapper.
- We define the data types of input and output key/value pair after the class declaration using angle brackets as done for Mapper.
- Both the input and the output of the Reducer is a key value pair.

Input:

- The key nothing but those unique words which have been generated after the sorting and shuffling phase: Text
- The value is a list of integers corresponding to each key: IntWritable
- Example – Bear, [1, 1], etc.

Output:

- The key is all the unique words present in the input text file: Text
- The value is the number of occurrences of each of the unique words: IntWritable
- Example – Bear, 2; Car, 3, etc.

- We have aggregated the values present in each of the list corresponding to each key and produced the final answer.
- In general, a single reducer is created for each of the unique words, but, you can specify the number of reducer in mapred-site.xml.

STEP 3: DRIVER CODE:

- In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
- We specify the name of the job , the data type of input/ output of the mapper and reducer.
- We also specify the names of the mapper and reducer classes.
- The path of the input and output folder is also specified.
- The method setInputFormatClass () is used for specifying that how a Mapper will read the input data or what will be the unit of work. Here, we have chosen TextInputFormat so that single line is read by the mapper at a time from the input text file. The main () method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.

PROGRAM:

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;
public class WordCount
{
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable>
{ public void map(LongWritable key, Text value,Context context) throws
IOException,InterruptedException{
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens())
{ value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
}
}
public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable>
{ public void reduce(Text key, Iterable<IntWritable> values,Context context)
throws IOException,InterruptedException {
int sum=0;

for(IntWritable x: values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}
}
public static void main(String[] args) throws Exception
{ Configuration conf= new Configuration();
```

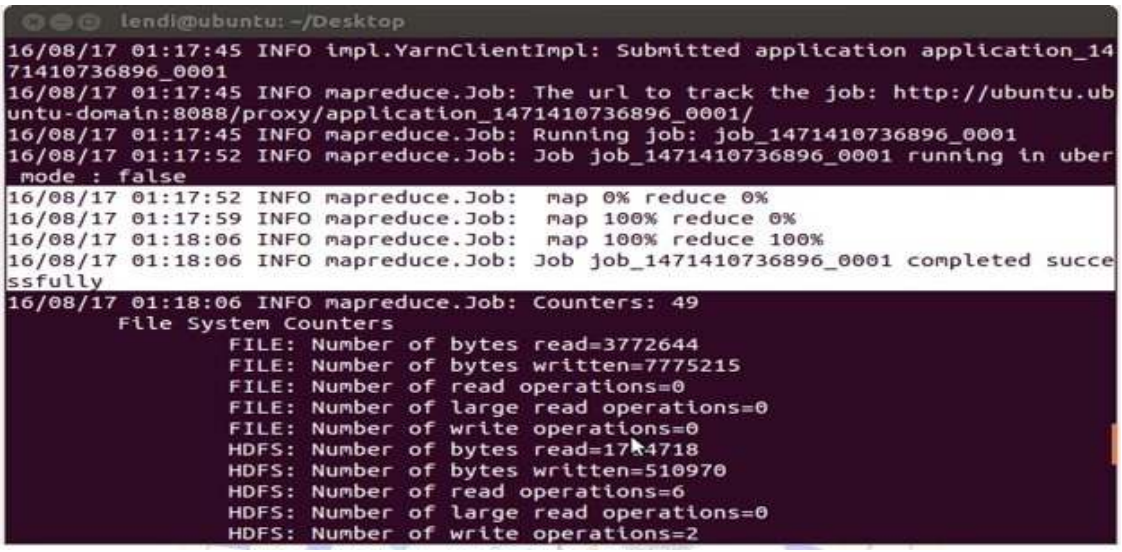
```
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
//deleting the output path automatically from hdfs so that we don't have to
delete it explicitly
outputPath.getFileSystem(conf).delete(outputPath);
//exiting the job only if the flag value becomes false
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

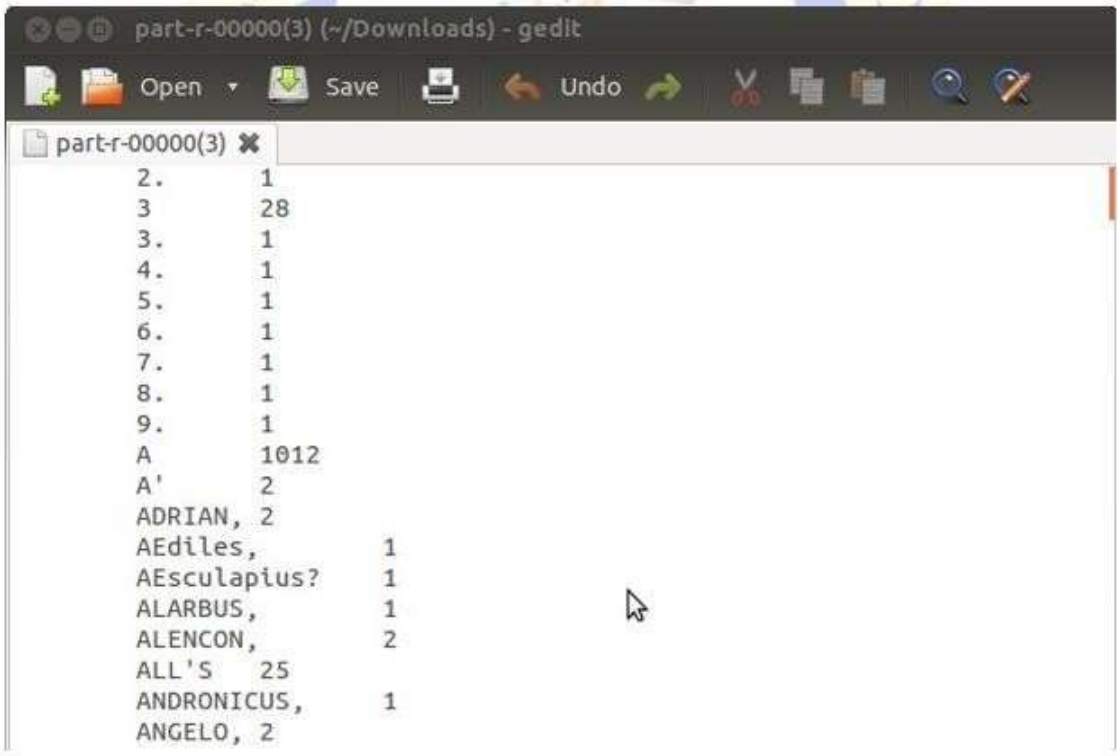
Run the MapReduce code:

The command for running a MapReduce code is:

hadoop jar hadoop-mapreduce-example.jar WordCount /sample/input /sample/output

OUTPUT:





RESULT:

Thus the Map Reduce Program that implements word count was executed and verified successfully.

EXP.NO:5 DATE:	INSTALLATION OF HIVE ALONG WITH PRACTICE EXAMPLES.

AIM:

To install HIVE along with practice examples.

PREREQUISITES:

- Java Development Kit (JDK) installed and the JAVA_HOME environment variable set.
- Hadoop installed and configured on your Windows system.

STEP-BY-STEP INSTALLATION:

1. **Download HIVE:**

   Visit the Apache Hive website and download the latest stable version of Hive.

   Official Apache Hive website: **https://hive.apache.org/**

2. **Extract the Downloaded Hive Archive to a Directory on Your Windows Machine,**

   e.g., C:\hive.

3. **Configure Hive:**

   - Open the Hive configuration file (hive-site.xml) located in the conf folder of the extracted Hive directory.
   - Set the necessary configurations, such as Hive Metastore connection settings and Hadoop configurations. Make sure to adjust paths accordingly for Windows. Here's an example of some configurations:

```
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:derby:;databaseName=/path/to/metastore_db;create=true
    </value>
    <description>JDBC connect string for a JDBC
    metastore.</description>
```

4. **Environment Variables Setup:**

   - Add the Hive binary directory (C:\hive\bin in this example) to your PATH environment variable.
   - Set the HIVE_HOME environment variable to point to the Hive installation directory (C:\hive in this example).

5. Start the Hive Metastore service:

To start the Hive Metastore service, you can use the schematool script:

6. Start Hive:

- Open a command prompt or terminal and navigate to the Hive installation directory.
- Execute the hive command to start the Hive shell.

EXAMPLES:

1. **Create a Database:**

   To create a new database in HIVE, use the following syntax:

   CREATE DATABASE database_name;

Example:

   CREATE DATABASE mydatabase;

2. **Use a Database:**

   To use a specific database in HIVE, use the following syntax:

   USE database_name;

Example:

   USE mydatabase;

3. **Show Databases:**

   To display a list of available databases in HIVE, use the following syntax:

   SHOW DATABASES;

4. **Create a Table:**

   To create a table in HIVE, use the following syntax:

```
CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  ...
);
```

Example:

```
CREATE TABLE mytable
  ( id INT,
  name STRING,
  age INT
);
```

5. **Show Tables:**

To display a list of tables in the current database, use the following syntax:

```
SHOW TABLES;
```

6. **Describe a Table:**

To view the schema and details of a specific table, use the following syntax:

```
DESCRIBE table_name;
```

Example:

```
DESCRIBE mytable;
```

7. **Insert Data into a Table:**

To insert data into a table in HIVE, use the following syntax:

```
INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);
```

Example:

```
INSERT INTO mytable (id, name, age) VALUES (1, 'John Doe', 25);
```

8. **Select Data from a Table:**

```
SELECT column1, column2, ... FROM table_name WHERE condition;
```

Example:

```
SELECT * FROM mytable WHERE age > 20;
```

RESULT:

Thus the Installation of HIVE was done successfully.

EXP.NO:6 DATE:	INSTALLATION OF HBASE ALONG WITH PRACTICE EXAMPLES

AIM:

To install HBASE using Virtual Machine and perform some operations in HBASE.

ALGORITHM:

Step 1: Install a Virtual Machine

- Download and install a virtual machine software such as VirtualBox (https://www.virtualbox.org/) or VMware (https://www.vmware.com/).
- Create a new virtual machine and install a Unix-based operating system like Ubuntu or CentOS. You can download the ISO image of your desired Linux distribution from their official websites.

Step 2: Set up the Virtual Machine

- Launch the virtual machine and install the Unix-based operating system following the installation wizard.
- Make sure the virtual machine has network connectivity to download software packages.

Step 3: Install Java

- Open the terminal or command line in the virtual machine.
- Update the package list

  sudo apt update
- Install OpenJDK (Java Development Kit):

  sudo apt install default-jdk
- Verify the Java installation:

  java -version

Step 4: Download and Install HBase

- In the virtual machine, navigate to the directory where you want to install HBase.
- Download the HBase binary distribution from the Apache HBase website (https://hbase.apache.org/). Look for the latest stable version.
- Extract the downloaded archive

  tar -xvf <hbase_archive_name>.tar.gz
- Replace <hbase_archive_name> with the actual name of the HBase archive file.

- Move the extracted HBase directory to a desired location:
  sudo mv <hbase_extracted_directory> /opt/hbase
- Replace <hbase_extracted_directory> with the actual name of the extracted HBase directory.

Step 5: Configure HBase

- Open the HBase configuration file for editing:
  sudo nano /opt/hbase/conf/hbase-site.xml
- Add the following properties to the configuration file:

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///var/lib/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/var/lib/zookeeper</value>
  </property>
</configuration>
```

- Save the file and exit the text editor.

Step 6: Start HBase

- Start the HBase server:
  sudo /opt/hbase/bin/start-hbase.sh

HBASE PRACTICE EXAMPLES:

Step 1: Start HBase

- Make sure HBase is installed and running on your Windows system.

Step 2: Open HBase Shell

- Open a command prompt or terminal window and navigate to the directory where the HBase installation is located. Run the following command to start the HBase shell:

>>hbase shell

Step 3: Create a Table

- In the HBase shell, you can create a table with column families.
- For example, let's create a table named "my_table" with a column family called "cf":

>> create 'my_table', 'cf'

Step 4: Insert Data

- To insert data into the table, you can use the put command.
- Here's an example of inserting a row with a specific row key and values:

>> put 'my_table', 'row1', 'cf:column1', 'value1'

>> put 'my_table', 'row1', 'cf:column2', 'value2'

Step 5: Get Data

- You can retrieve data from the table using the get command.
- For example, to get the values of a specific row:

>> get 'my_table', 'row1'

- This will display all the column family values for the specified row.

Step 6: Scan Data

- To scan and retrieve multiple rows or the entire table, use the scan command.
- For instance, to scan all rows in the table:

>> scan 'my_table'

- This will display all rows and their corresponding column family values.

Step 7: Delete Data

- To delete a specific row or a particular cell value, you can use the delete command.
- Here's an example of deleting a specific row:

>>delete 'my_table', 'row1'

Step 8: Disable and Drop Table

- If you want to remove the table entirely, you need to disable and drop it.
- Use the following commands:

>>disable 'my_table'

>>drop 'my_table'

RESULT:

Thus the installation of HBase using Virtual Machine was done successfully.

EXP.NO:7	INSTALLATION OF THRIFT
DATE:	

AIM:

To install Apache thrift on Windows OS.

ALGORITHM:

Step 1: Download Apache Thrift:

- Visit the Apache Thrift website: https://thrift.apache.org/
- Go to the "Downloads" section and find the latest version of Thrift.
- Download the Windows binary distribution (ZIP file) for the desired version.

Step 2: Extract the ZIP file:

- Locate the downloaded ZIP file and extract its contents to a directory of your choice.
- This directory will be referred to as <THRIFT_DIR> in the following steps.

Step 3: Set up environment variables:

- Open the Start menu and search for "Environment Variables" and select "Edit the system environment variables."
- Click the "Environment Variables" button at the bottom right of the "System Properties" window.
- Under the "System variables" section, find the "Path" variable and click "Edit."
- Add the following entries to the "Variable value" field (replace <THRIFT_DIR> with the actual directory path):

  <THRIFT_DIR>\bin

  <THRIFT_DIR>\lib

- Click "OK" to save the changes.

Step 4: Verify the installation:

- Open a new Command Prompt window.
- Run the following command to verify that Thrift is installed and accessible:

  thrift –version

- If everything is set up correctly, you should see the version number of Thrift printed on the screen.

RESULT:

Thus the installation of Thrift on windows OS was done successfully.

EXP.NO:8 DATE:	PRACTICE IMPORTING AND EXPORTING DATA FROM VARIOUS DATABASES.

AIM:

To import and export data from various Databases using SQOOP.

ALGORITHM:

Step 1: Install SQOOP.

- First, you need to install Sqoop on your Hadoop cluster or machine.
- Download the latest version of Sqoop from the Apache Sqoop website (http://sqoop.apache.org/) and follow the installation instructions provided in the documentation

Step 2: Importing data from a database:

- To import data from a database into Hadoop, use the following Sqoop command:

```
Sqoop import –connect
jdbc:<DB_TYPE>://<DB_HOST>:<DB_PORT>/<DB_NAME> \
        --username <DB_USERNAME> \
        --password <DB_PASSWORD> \
        --table <TABLE_NAME> \
        --target-dir <HDFS_TARGET_DIR> \
        --m <NUMBER_OF_MAP_TASKS>
```

- Replace the placeholders
- (<DB_TYPE>, <DB_HOST>, <DB_PORT>, <DB_NAME>, <DB_USERNAME>, <DB_PASSWORD>, <TABLE_NAME>, <HDFS_TARGET_DIR>, and <NUMBER_OF_MAP_TASKS>) with the appropriate values for your database and Hadoop environment.

Step 3: Exporting data to a database:

To export data from Hadoop to a database, use the following Sqoop command:

```
sqoop export –connect
jdbc:<DB_TYPE>://<DB_HOST>:<DB_PORT>/<DB_NAME> \
        --username <DB_USERNAME> \
```

```
--password <DB_PASSWORD> \
--table <TABLE_NAME> \
--export-dir <HDFS_EXPORT_DIR> \
--input-fields-terminated-by '<DELIMITER>'
```

- Replace the placeholders
- (<DB_TYPE>, <DB_HOST>, <DB_PORT>, <DB_NAME>, <DB_USERNAME>, <DB_PASSWORD>, <TABLE_NAME>, <HDFS_EXPORT_DIR>, and <DELIMITER>) with the appropriate values for your database and Hadoop environment.

RESULT:

Thus the implementation export data from various Databases using SQOOP was done successfully.