EX.NO: **DOWNLOADING AND INSTALLING HADOOP; UNDERSTANDING DIFFERENT HADOOP MODES, STARTUP SCRIPTS, CONFIGURATION FILES**

DATE:

AIM:

To download and install Hadoop, understand different Hadoop modes, startup scripts, configuration files.

PROCEDURE:

Step 1 – Installing Java

Check if java is already installed on your system using the command

```
$ java -version
```

```
openjdk version "11.0.11" 2021-04-20
OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.20.04)
OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.20.04, mixed mode, sharing)
```

If not, install OpenJDK 11 from the default apt repositories:

```
$ sudo apt update
$ sudo apt install openjdk-11-jdk
```

Step 2 – Create a Hadoop User

Run the following command to create a new user with name hadoop:

```
$ sudo adduser hadoop
```

Provide and confirm the new password as shown below:

```
Adding user `hadoop' ...
Adding new group `hadoop' (1002) ...
Adding new user `hadoop' (1002) with group `hadoop' ...
Creating home directory `/home/hadoop' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for hadoop
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
```

Step 3 – Configure SSH Key-based Authentication

Change the user to hadoop with the following command:

```
$ su - hadoop
```

Run the following command to generate Public and Private Key Pairs:

```
$ ssh-keygen -t rsa
```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:QSa2syeISwP0hD+UXxxi0j9MSOrjKDGIbkfbM3ejyIk hadoop@ubuntu20
The key's randomart image is:
+---[RSA 3072]----+
| ..o++=.+        |
|..oo++.O         |
|. oo. B .        |
|o..+ o * .       |
|= ++o o S        |
|.++o+  o         |
|.+.+ + . o       |
|o . o * o .      |
|   E + .         |
+----[SHA256]-----+
```

Next, append the generated public keys from id_rsa.pub to authorized_keys and set proper permission:

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 640 ~/.ssh/authorized_keys
```

Verify the passwordless SSH authentication with the following command:

```
$ ssh localhost
```

Authenticate hosts by adding RSA keys to known hosts. Type yes and hit Enter to authenticate the localhost:

```
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:JFqDVbM3zTPhUPgD5oMJ4ClviH6tzIRZ2GD3BdNqGMQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

Step 4 – Installing Hadoop

Change the user to hadoop with the following command:

```
$ su - hadoop
```

Download the latest version of Hadoop using the wget command:

```
$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
```

Once downloaded, extract the downloaded file:

```
$ tar -xvzf hadoop-3.3.0.tar.gz
```

Rename the extracted directory to hadoop:

```
$ mv hadoop-3.3.0 hadoop
```

To configure Hadoop and Java Environment Variables on your system, Open the ~/.bashrc file in text editor:

```
$ nano ~/.bashrc
```

Append the below lines to file.

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

Save and close the file. Then, activate the environment variables with the following command:

```
$ source ~/.bashrc
```

Open the Hadoop environment variable file:

```
$ nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

Again set the JAVA_HOME in the Hadoop environment.

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

Save and close the file.

Step 5 – Configuring Hadoop

To create the namenode and datanode directories inside Hadoop home directory, run the following command to create both directories:

```
$ mkdir -p ~/hadoopdata/hdfs/namenode
$ mkdir -p ~/hadoopdata/hdfs/datanode
```

Edit the **core-site.xml** file and update with system hostname:

```
$ nano $HADOOP_HOME/etc/hadoop/core-site.xml
```

Change the following name as per system hostname:

```
<configuration>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://hadoop.tecadmin.com:9000</value>
        </property>
</configuration>
```

Save and close the file. Then, edit the **hdfs-site.xml** file:

```
$ nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

Change the NameNode and DataNode directory path:

```
<configuration>

        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>

        <property>
                <name>dfs.name.dir</name>
                <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
        </property>

        <property>
                <name>dfs.data.dir</name>
                <value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
        </property>
</configuration>
```

Save and close the file. Then, edit the **mapred-site.xml** file:

```
$ nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

Make the following changes:

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

Save and close the file. Then, edit the **yarn-site.xml** file:

```
$ nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

Make the following changes:

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

Save and close the file.

Step 6 – Start Hadoop Cluster

Run the following command to format the Hadoop Namenode:

```
$ hdfs namenode -format
```

```
2020-11-23 10:31:51,318 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with
2020-11-23 10:31:51,323 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet
2020-11-23 10:31:51,323 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoop.tecadmin.net/127.0.1.1
************************************************************/
```

After formatting the Namenode, run the following command to start the Hadoop cluster:

```
$ start-dfs.sh
```

```
Starting namenodes on [hadoop.tecadmin.com]
hadoop.tecadmin.com: Warning: Permanently added 'hadoop.tecadmin.com,fe80::200:2dff:fe3a:26ca%e
Starting datanodes
Starting secondary namenodes [hadoop.tecadmin.com]
```

Next, start the YARN service:

```
$ start-yarn.sh
```

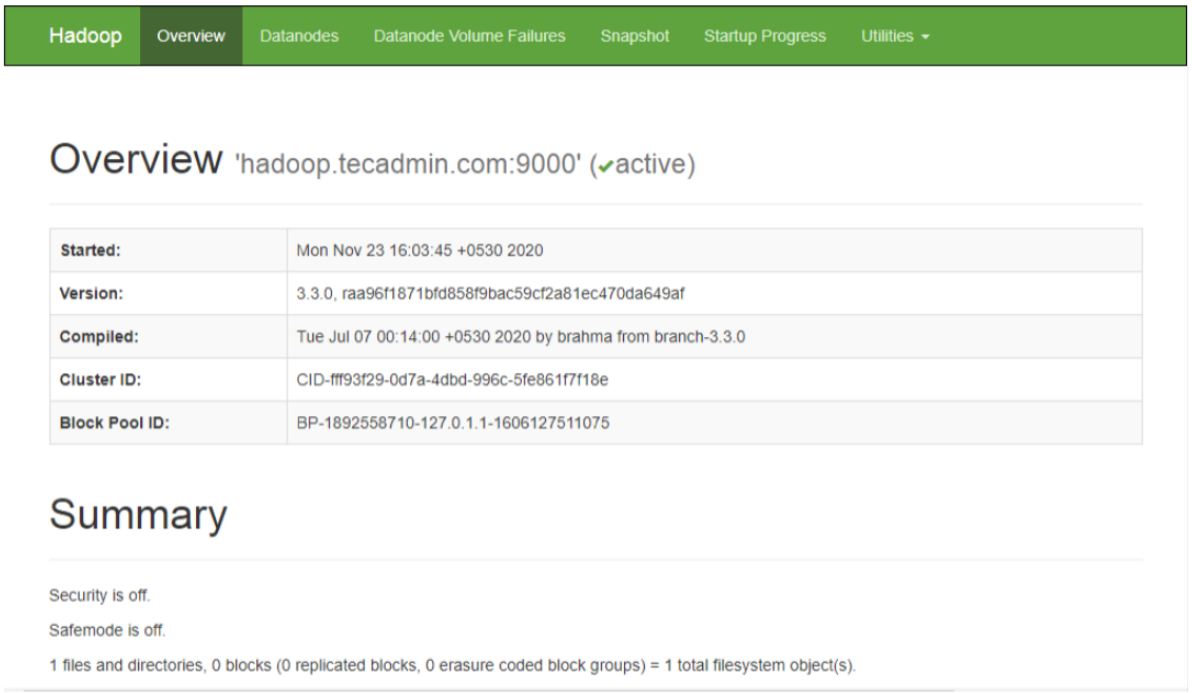
```
Starting resourcemanager
Starting nodemanagers
```

Check the status of all Hadoop services using the jps command:

```
$ jps
```

All the running services are seen in the following output:

```
18194 NameNode
18822 NodeManager
17911 SecondaryNameNode
17720 DataNode
18669 ResourceManager
19151 Jps
```



Step 7– Stop Hadoop Cluster

To stop the Hadoop Namenode service, run the following command as a hadoop user:

```
$ stop-dfs.sh
```

To stop the Hadoop Resource Manager service, run the following command:

```
$ stop-yarn.sh
```

RESULT:

Thus the above experiment to install and configure Hadoop was successfully executed and verified.

EX.NO:

HADOOP IMPLEMENTATION OF FILE MANAGEMENT TASKS

DATE:

Aim:

To implement Hadoop file management tasks, such as Adding files and directories, retrieving files, and deleting files

Steps:

1) Start the distributed file system and follow the command below to start the namenode as well as the data nodes in cluster.

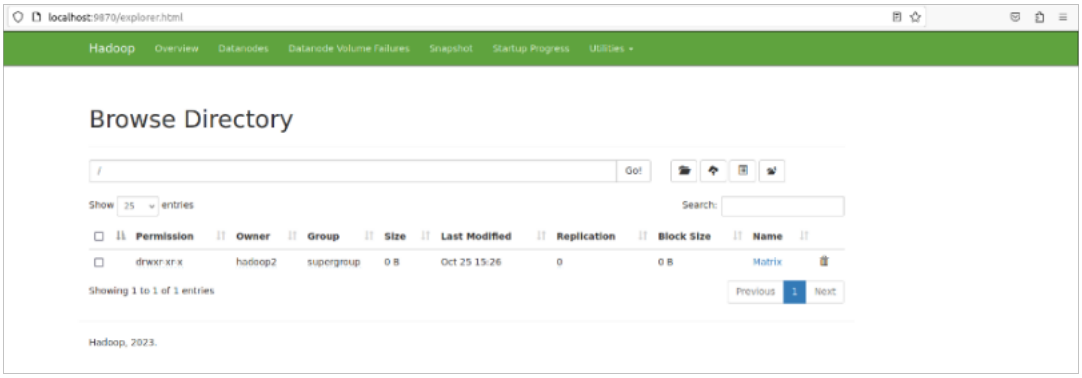
$ start-dfs.sh

```
hadoop1@cclab-HP-280-G2-MT-Legacy:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [cclab-HP-280-G2-MT-Legacy]
```

2) Create a directory in HDFS at given path using the command

hadoop fs -mkdir <path>

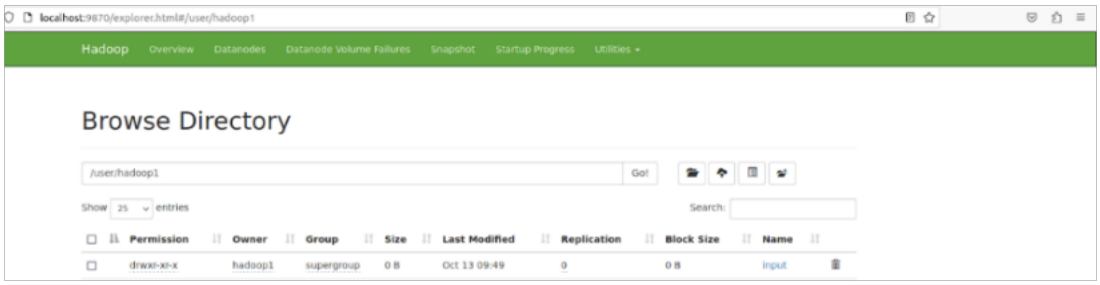
```
hadoop2@cloudlab54-HP-280-G2-MT-Legacy:~$ hadoop fs -mkdir /Matrix
hadoop2@cloudlab54-HP-280-G2-MT-Legacy:~$
```



3) Copy single src file, or multiple src files from local file system to the Hadoop data file system using the command

hadoop fs -put <localsrc>...<HDFS_path>

Eg: hadoop fs -put '/home/hadoop/Desktop/Matrix/input' /Matrix



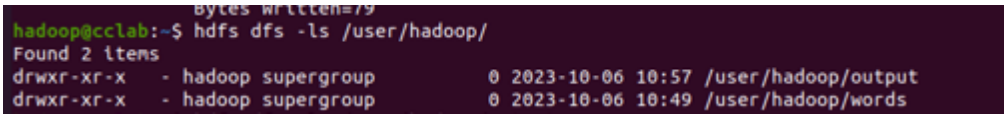
4) Similarly we can copy a file from hadoop to our local system using the

command

hadoop fs -get <hdfs_src> <local_dst>

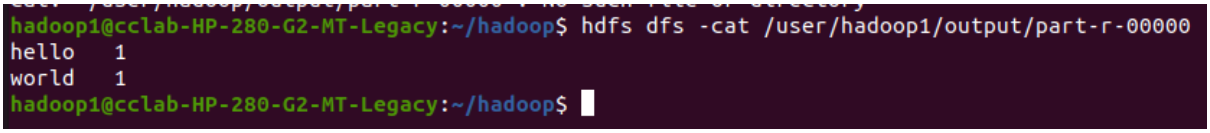
5) List the contents of a directory using the command

hadoop fs -ls <args>



6) To see the content of a file the following command is used

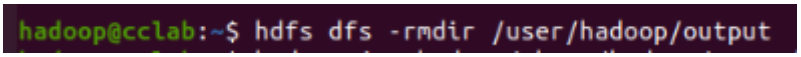
hadoop fs -cat <filepath>

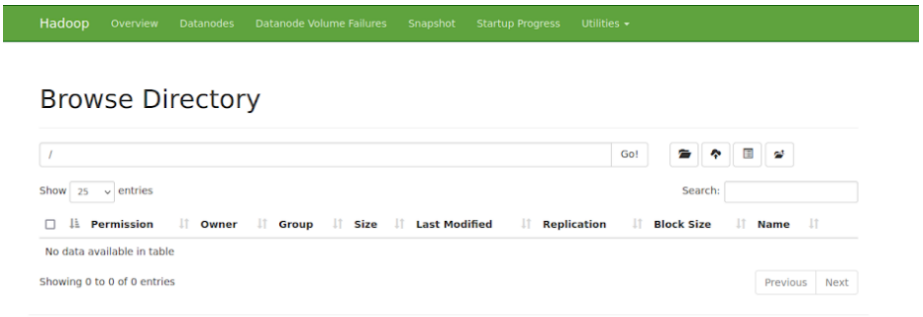


7) To remove a file or directory from HDFS , use the following command

hadoop fs -rm <arg> (for file)

hadoop fs -rmdir <arg> (for directory)





Result:

Thus the above experiment to implement Hadoop file management task was successfully executed and verified.

Source code:

WC_Mapper.java

```
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.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;
```

```
import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class WC_Mapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{

   private final static IntWritable one = new IntWritable(1);

   private Text word = new Text();

   public void map(LongWritable key, Text
value,OutputCollector<Text,IntWritable> output,

         Reporter reporter) throws IOException{

      String line = value.toString();

      StringTokenizer  tokenizer = new StringTokenizer(line);

      while (tokenizer.hasMoreTokens()){

         word.set(tokenizer.nextToken());

         output.collect(word, one);

      }

   }   }
```

WC_Reducer.java

```
import java.io.IOException;

   import java.util.Iterator;

   import org.apache.hadoop.io.IntWritable;

   import org.apache.hadoop.io.Text;

   import org.apache.hadoop.mapred.MapReduceBase;

   import org.apache.hadoop.mapred.OutputCollector;

   import org.apache.hadoop.mapred.Reducer;

   import org.apache.hadoop.mapred.Reporter;
```

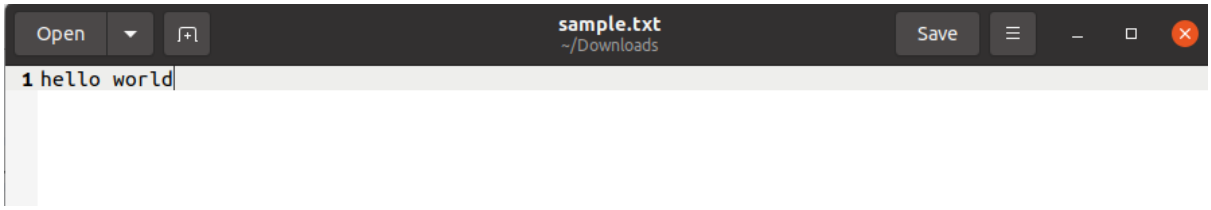
```
   public class WC_Reducer  extends MapReduceBase implements
Reducer<Text,IntWritable,Text,IntWritable> {

   public void reduce(Text key, Iterator<IntWritable>
values,OutputCollector<Text,IntWritable> output,

    Reporter reporter) throws IOException {

   int sum=0;

   while (values.hasNext()) {

   sum+=values.next().get();

   }

   output.collect(key,new IntWritable(sum));

   }

   }
```

WC_Runner.java

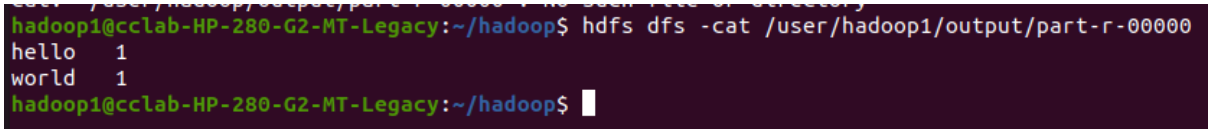
```
import java.io.IOException;

   import org.apache.hadoop.fs.Path;

   import org.apache.hadoop.io.IntWritable;

   import org.apache.hadoop.io.Text;

   import org.apache.hadoop.mapred.FileInputFormat;

   import org.apache.hadoop.mapred.FileOutputFormat;

   import org.apache.hadoop.mapred.JobClient;

   import org.apache.hadoop.mapred.JobConf;

   import org.apache.hadoop.mapred.TextInputFormat;

   import org.apache.hadoop.mapred.TextOutputFormat;

   public class WC_Runner {

      public static void main(String[] args) throws IOException{
```

```
        JobConf conf = new JobConf(WC_Runner.class);
        conf.setJobName("WordCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(WC_Mapper.class);
        conf.setCombinerClass(WC_Reducer.class);
        conf.setReducerClass(WC_Reducer.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf,new Path(args[0]));
        FileOutputFormat.setOutputPath(conf,new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

Input file



Output:



Source code:

Map.java

```
package matrix_multiplication;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class Map extends
org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, Text>
```

```
{       @Override
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
                Configuration conf = context.getConfiguration();
                int m = Integer.parseInt(conf.get("m"));
                int p = Integer.parseInt(conf.get("p"));
String line = value.toString();
// (M, i, j, Mij);
String[] indicesAndValue = line.split(",");
Text outputKey = new Text();
Text outputValue = new Text();
if (indicesAndValue[0].equals("M")) {
for (int k = 0; k < p; k++) {
outputKey.set(indicesAndValue[1] + "," + k);
// outputKey.set(i,k);
outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2]
+ "," + indicesAndValue[3]);
// outputValue.set(M,j,Mij);
context.write(outputKey, outputValue);
}
} else {
        // (N, j, k, Njk);
        for (int i = 0; i < m; i++) {
        outputKey.set(i + "," + indicesAndValue[2]); outputValue.set("N," +
indicesAndValue[1] + ","
```

```
        + indicesAndValue[3]); context.write(outputKey, outputValue);
        }
        }
        }
}
```

MapReduce.java

```
package matrix_multiplication;

import org.apache.hadoop.io.Text;

//import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

import java.util.HashMap;

public class Reduce

extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text>
{ @Override

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

String[] value;

//key=(i,k),

//Values = [(M/N,j,V/W),..]

HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
HashMap<Integer, Float> hashB = new HashMap<Integer, Float>(); for (Text
val : values) {

value = val.toString().split(",");

if (value[0].equals("M")) {

hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2])); } else {

hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
```

```
}

}

int n = Integer.parseInt(context.getConfiguration().get("n"));

float result = 0.0f;

float m_ij;

float n_jk;

for (int j = 0; j < n; j++) {

m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f; n_jk = hashB.containsKey(j) ?
hashB.get(j) : 0.0f; result += m_ij * n_jk;

}

if (result != 0.0f) {

context.write(null,new Text(key.toString() + "," + Float.toString(result)));

}

}

}
```

MatrixMultiply.java

```
package matrix_multiplication;


import org.apache.hadoop.conf.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class MatrixMultiply {
public static void main(String[] args) throws Exception { if (args.length != 2) {
System.err.println("Usage: MatrixMultiply <in_dir> <out_dir>");
System.exit(2);
}
Configuration conf = new Configuration();
conf.set("m", "1000");
conf.set("n", "100");
conf.set("p", "1000");
@SuppressWarnings("deprecation")
Job job = new Job(conf, "MatrixMultiply");
job.setJarByClass(MatrixMultiply.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);
}
```

}

Steps to run the program

javac -version

hadoop version

export HADOOP_CLASSPATH=$(hadoop classpath)

echo $HADOOP_CLASSPATH

1. Loading Input Files to HDFS

hadoop fs -mkdir /Matrix

hadoop fs -mkdir /Matrix/Input

hadoop fs -put '/home/hadoop/Desktop/Matrix/input/M.txt'
'/home/hadoop/Desktop/Matrix/input/N.txt' /Matrix/Input

2. Creating JAR Files

cd /home/hadoop/Desktop/Matrix

javac -classpath ${HADOOP_CLASSPATH} -d
'/home/hadoop/Desktop/Matrix/tutorial_classes2'
'/home/hadoop/Desktop/Matrix/MatrixMultiply.java'
'/home/hadoop/Desktop/Matrix/Map.java'
'/home/hadoop/Desktop/Matrix/Reduce.java'

jar -cvf firstMatrix.jar -C tutorial_classes2/ .

3. Executing Matrix Multiplication in Hadoop

hadoop jar '/home/hadoop/Desktop/Matrix/firstMatrix.jar'

matrix_multiplication/MatrixMultiply /Matrix1/Input1/ /Matrix1/Output1

Input files:

M.txt

```
M,0,0,1
M,0,1,2
M,1,0,3
M,1,1,4
```

N.txt

```
N,0,0,5
N,0,1,6
N,1,0,7
N,1,1,8
```

Output:

```
hadoop@cloudlab54-HP-280-G2-MT-Legacy:~/Desktop/Matrix$ hadoop fs -cat /Matrix/Output/part-r-00000
0,0,19.0
0,1,22.0
1,0,43.0
1,1,50.0
```