SHREENIVASA ENGINEERING COLLEGE

(Approved by AICTE and Affiliated to Anna University,Chennai)

B.Pallipatti, Bommidi, Pappireddipatti (Tk), Dharmapuri Dt, Tamilnadu-635 301

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE



Bonafide Certificate

This is to certify that this is a Bonafide Record of Practical work done by……of .....Semester of ..... Department in the ……Laboratory During the year……..

Head of the Department Staff Incharge

Submitted for the University Examination held on …….

Internal Examiner External Examiner

S.No.	Date	Name of the Experiment	Page No.	Marks	Signature

S.No.	Date	Name of the Experiment	Page No.	Marks	Signature

Ex.No:1	HADOOP INSTALLATION
Date:	

AIM

To Download and install Hadoop and Understanding different Hadoop modes, Startup scripts, Configuration files.

PROCEDURE:

Step by step Hadoop 2.8.0 installation on Windows 10 Prepare:

These software‟s should be prepared to install Hadoop 2.8.0 on window 10 64 bits.

1) Download Hadoop 2.8.0
   (Link: http://wwweu.apache.org/dist/hadoop/common/hadoop-2.8.0/hadoop-2.8.0.tar.gz OR http://archive.apache.org/dist/hadoop/core//hadoop-2.8.0/hadoop-2.8.0.tar.gz)
2) Java JDK 1.8.0.zip
   (Link: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)

Set up:

1) Check either Java 1.8.0 is already installed on your system or not, use “Javac -version" to check Java version

2) If Java is not installed on your system then first install java under "C:\JAVA" Javasetup

3) Extract files Hadoop 2.8.0.tar.gz or Hadoop-2.8.0.zip and place under "C:\Hadoop-2.8.0" hadoop

4) Set the path HADOOP_HOME Environment variable on windows 10(see Step 1,2, 3 and 4 below) hadoop

5) Set the path JAVA_HOME Environment variable on windows 10(see Step 1, 2,3 and 4 below) java

6) Next we set the Hadoop bin directory path and JAVA bin directory path

Configuration

a) File C:/Hadoop-2.8.0/etc/hadoop/core-site.xml, paste below xml paragraph andsave this file.

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

b) Rename "mapred-site.xml.template" to "mapred-site.xml" and edit this file C:/Hadoop-2.8.0/etc/hadoop/mapred-site.xml, paste below xml paragraph and savethis file.

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

c) Create folder "data" under "C:\Hadoop-2.8.0"

1) Create folder "datanode" under "C:\Hadoop-2.8.0\data"

2) Create folder "namenode" under "C:\Hadoop-2.8.0\data" data

d) Edit file C:\Hadoop-2.8.0/etc/hadoop/hdfs-site.xml, paste below xml paragraphand save this file.

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

```
    <name>dfs.namenode.name.dir</name>
    <value>C:\hadoop-2.8.0\data\namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>C:\hadoop-2.8.0\data\datanode</value>
  </property>
</configuration>
```

e) Edit file C:/Hadoop-2.8.0/etc/hadoop/yarn-site.xml, paste below xml paragraphand 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>
```

f) Edit file C:/Hadoop-2.8.0/etc/hadoop/hadoop-env.cmd by closing the command line "JAVA_HOME=%JAVA_HOME%" instead of set "JAVA_HOME=C:\Java" (On C:\java this is path to file jdk.18.0)

Hadoop Configuration

7) Download file Hadoop Configuration.zip (Link:https://github.com/MuhammadBilalYar/HADOOP-INSTALLATION-ON-WINDOW-10/blob/master/Hadoop%20Configuration.zip)

8) Delete file bin on C:\Hadoop-2.8.0\bin, replaced by file bin on file just download(from Hadoop Configuration.zip).

9) Open cmd and typing command "hdfs namenode –format" .You will see hdfs namenode –format

Testing

10) Open cmd and change directory to "C:\Hadoop-2.8.0\sbin" and type "start-all.cmd" to start apache.

11) Make sure these apps are running.

a) Name node

b) Hadoop data node

c) YARN Resource Manager

d) YARN Node Manager hadoop nodes

12) Open: http://localhost:8088

13) Open: http://localhost:50070

RESULT

Installing Hadoop and Understanding different Hadoop modes. Startup scripts, Configuration files was executed successfully.

Ex.No:2	HADOOP IMPLEMENTATION OF FILE MANAGEMENT TASKS
Date:	

AIM:

To write a program to implement Hadoop Implementation of file management tasks

PROGRAM :

Implement the following file management tasks in Hadoop:

i. Adding files and directories

ii. Retrieving files

iii. Deleting files

Adding Files and Directories to HDFS

Before you can run Hadoop programs on data stored in HDFS, you,,ll need to put the data into HDFS first . Let,,s create a directory and put a file in it. HDFS has a default working directory of /user/$USER, where $USER is your login user name. This directory isn,,t automatically created for you, though, so let,,s create it with the mkdir command. For the purpose of illustration, we use chuck. You should substitute your user name in the example commands.

hadoop fs -mkdir /user/chuck

hadoop fs -put example.txt /user/chuck

Retrieving Files from HDFS

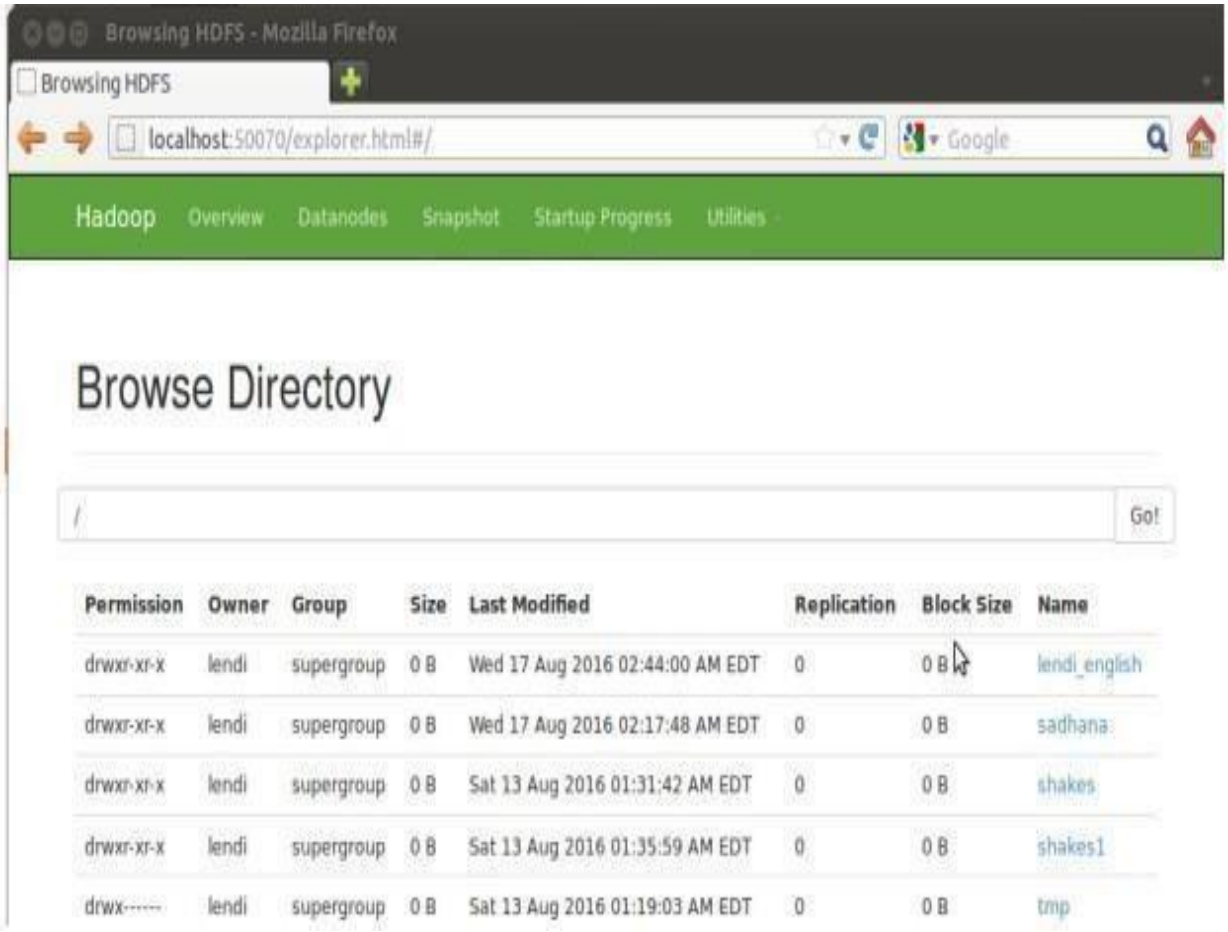
The Hadoop command get copies files from HDFS back to the local filesystem. To retrieve example.txt,

we can run the following command:

hadoop fs -cat example.txt

Deleting files from HDFS

hadoop fs -rm example.txt


Browsing HDFS - Mozilla Firefox
Browsing HDFS
localhost:50070/explorer.html#/
Google
Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities
Browse Directory
/
Go!
Permission Owner Group Size Last Modified Replication Block Size Name
drwxr-xr-x lendi supergroup 0 B Wed 17 Aug 2016 02:44:00 AM EDT 0 0 B lendi_english
drwxr-xr-x lendi supergroup 0 B Wed 17 Aug 2016 02:17:48 AM EDT 0 0 B sadhana
drwxr-xr-x lendi supergroup 0 B Sat 13 Aug 2016 01:31:42 AM EDT 0 0 B shakes
drwxr-xr-x lendi supergroup 0 B Sat 13 Aug 2016 01:35:59 AM EDT 0 0 B shakes1
drwx------ lendi supergroup 0 B Sat 13 Aug 2016 01:19:03 AM EDT 0 0 B tmp

- Command for creating a directory in hdfs is “hdfs dfs –mkdir /lendicse”.
- Adding directory is done through the command “hdfs dfs –put lendi_english /”.

RESULT

Thus, the program to implement Hadoop Implementation of file management tasks was executed and verified successfully.

Ex.No:3	IMPLEMENT OF MATRIX MULTIPLICATION WITH HADOOP MAP REDUCE
Date:	

AIM:

To Develop a Map Reduce program to implement Matrix Multiplication.

Procedure:

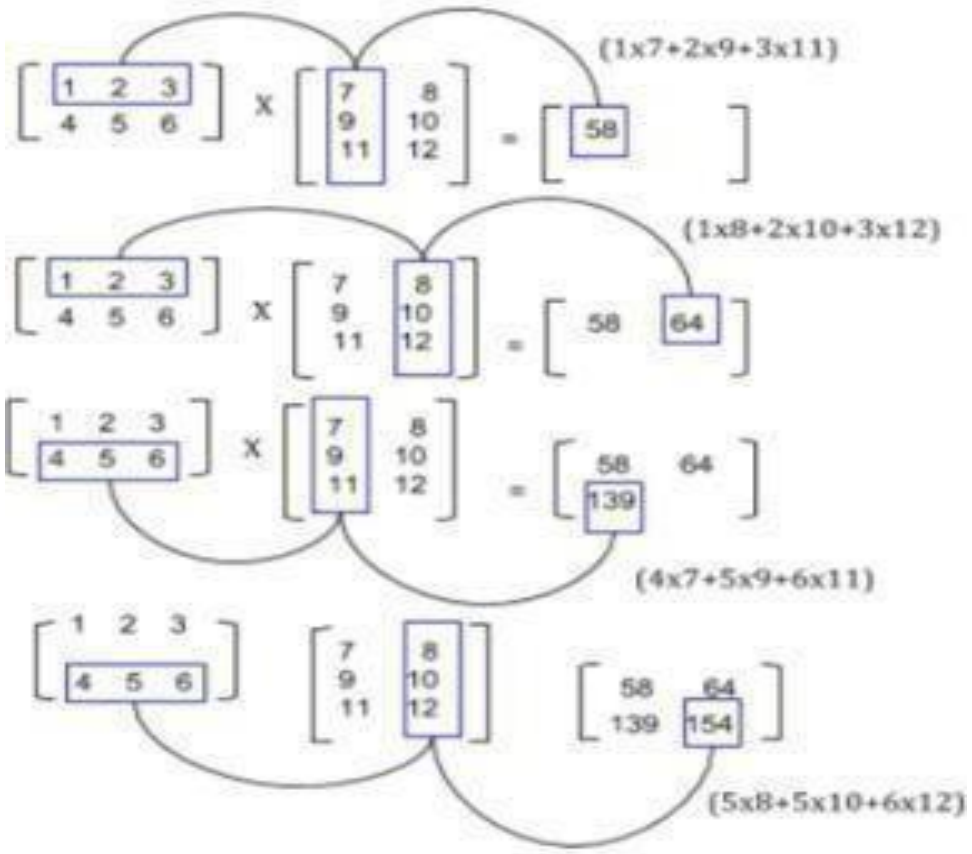
In mathematics, matrix multiplication or the matrix product is a binary operation that produces a matrix from two matrices. The definition is motivated by linear equations and linear transformations on vectors, which have numerous applications in applied mathematics, physics, and engineering. In more detail, if A is an $n \times m$ matrix and B is an $m \times p$ matrix, their matrix product AB is an $n \times p$ matrix, in which the m entries across a row of A are multiplied with the m entries down a column of B and summed to produce an entry of AB. When two linear transformations are represented by matrices, then the matrix product represents the composition of the two transformations.

- Create two files M1, M2 and put the matrix values. (sperate columns with spaces and rows with a line break)

```
For this example I am taking matrices as:
1 2 3       7  8
4 5 6       9  10
          11 12
```

- Put the above files to HDFS at location /user/clouders/matrices/

```
hdfs dfs -mkdir /user/cloudera/matrice
hdfs dfs -put /path/to/M1 /user/cloudera/matrices/
hdfs dfs -put /path/to/M2 /user/cloudera/matrices/
```



Algorithm for Map Function.

a. for each element m_{ij} of M do

produce (key,value) pairs as ((i,k), (M,j,m_{ij}), for k=1,2,3,.. upto the number of columns of N

b. for each element njk of N do
produce (key,value) pairs as ((i,k),(N,j,N_{jk}), for i = 1,2,3,.. Upto the number of rows of M.

c. return Set of (key,value) pairs that each key (i,k), has list with values (M,j,m_{ij}) and (N, j,n_{jk}) for all possible values of j.

Algorithm for Reduce Function.

d. for each key (i,k) do
e. sort values begin with M by j in listM sort values begin with N by j in listN multiply mij and njk for jth value of each list
f. sum up mij x njk return (i,k), Σj=1 mij x njk

Step 1. Download the hadoop jar files with these links.

Download Hadoop Common Jar files: https://goo.gl/G4MyHp
$ wget https://goo.gl/G4MyHp -O hadoop-common-2.2.0.jar
Download Hadoop Mapreduce Jar File: https://goo.gl/KT8yfB
$ wget https://goo.gl/KT8yfB -O hadoop-mapreduce-client-core-2.7.1.jar

Step 2. Creating Mapper file for Matrix Multiplication.

```
import java.io.DataInput;
import java.io.DataOutput;
import
java.io.IOException;import
java.util.ArrayList;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.ReflectionUtils;
class Element implements writable {
int tag;
int index;
double value;
        Element(){
                tag = 0;
                index = 0;
                value = 0.0;
        }
        Element(int tag, int index, double value)
        {
        this.tag = tag;
```

```
            this.index = index;
            this.value = value;
        }
        @Override
        public void readFields(DataInput input) throws
            IOException {tag = input.readInt();
            index = input.readInt();
            value = input.readDouble();
        }
        @Override
        public void write(DataOutput output) throws
            IOException {output.writeInt(tag);
            output.writeInt(index);
            output.writeDouble(value);
        }
 }
 class Pair implements WritableComparable<Pair>
 {
 int i;
int j;

        Pair() {
            i = 0;
            j = 0;
        }

        Pair(int i, int j)
        { this.i = i;
        this.j = j;

        }
        @Override
        public void readFields(DataInput input) throws
            IOException
         {
        i = input.readInt();
            j = input.readInt();
        }
```

```
        @Override
        public void write(DataOutput output) throws
                IOException {output.writeInt(i);
                output.writeInt(j);
        }
        @Override
        public int compareTo(Pair compare)
        {
        if (i > compare.i) {
                        return 1;
                } else if ( i <
                        compare.i)
                        {return -1;
                } else {
                        if(j > compare.j) {
                                return 1;
                        } else if (j <
                                compa
                                re.j)  {
                                return
                                -1;
                        }}
                return 0;
        }
        public String
                toString()
                {
        return i + " " + j + " ";
        }
}
public class Multiply {
        public static class MatriceMapperM extends Mapper<Object,Text,IntWritable,Element>
{
                @Override
                public void map(Object key, Text value, Context context) throws
                                IOException, InterruptedException
                 {
                        String readLine =
```

```
                value.toString(); String[]
                stringTokens =
                readLine.split(",");
                int index = Integer.parseInt(stringTokens[0]);
                double elementValue =
                Double.parseDouble(stringTokens[2]);
                Element e = new Element(0, index, elementValue);
                IntWritable keyValue = new
                IntWritable(Integer.parseInt(stringTokens[1]));
                context.write(keyValue, e);
            }
        }
        public static class MatriceMapperN extends
            Mapper<Object,Text,IntWritable,Element> {@Override
            public void map(Object key, Text value, Context context)
                    throws IOException, InterruptedException
            {
                String readLine = value.toString();
                String[] stringTokens = readLine.split(",");
                int index = Integer.parseInt(stringTokens[1]);
                double elementValue = Double.parseDouble(stringTokens[2]);
                Element e = new Element(1,index, elementValue);
                IntWritable keyValue = new
    IntWritable(Integer.parseInt(stringTokens[0]));
                context.write(keyValue, e);
            }
        }
        public static class ReducerMxN extends
    Reducer<IntWritable,Element, Pair,DoubleWritable>
         {
        @Override
        public void reduce(IntWritable key, Iterable<Element> values, Context
    context) throwsIOException, InterruptedException
        {
            ArrayList<Element> M = new ArrayList<Element>();
            ArrayList<Element> N = new ArrayList<Element>();Configuration conf
            = context.getConfiguration(); for(Element element : values) {
Element tempElement = ReflectionUtils.newInstance(Element.class, conf);
```

```
                ReflectionUtils.copy(conf, element, tempElement);
                if
                        (tempElem
                        ent.tag ==
                        0) {
                        M.add(tem
                        pElement);
                } else
                        if(tempElemen
                        t.tag == 1) {
                        N.add(tempEle
                        ment);
                }
        }

        for(int i=0;i<M.size();i++) {
                for(int j=0;j<N.size();j++) {

                        Pair p = new Pair(M.get(i).index,N.get(j).index);
                        double multiplyOutput = M.get(i).value * N.get(j).value;

                        context.write(p, new DoubleWritable(multiplyOutput));
                        }
                }
        }
}
public static class MapMxN extends Mapper<Object, Text, Pair,
        DoubleWritable>
 {
@Override
        public void map(Object key, Text value, Context context)
                        throws IOException, InterruptedException {
                String readLine = value.toString(); String[]
                pairValue = readLine.split(" ");
                Pair p = new
Pair(Integer.parseInt(pairValue[0]),Integer.parseInt(pairValue[1]));
                DoubleWritable val = new
DoubleWritable(Double.parseDouble(pairValue[2]));
                context.write(p, val);
        }
```

```
	}
	public static class ReduceMxN extends Reducer<Pair,
DoubleWritable, Pair,DoubleWritable>
	{
		@Override
		public void reduce(Pair key, Iterable<DoubleWritable> values,
Context context)throws IOException, InterruptedException
		{
			double sum = 0.0;
			for(DoubleWritable value : values) {
				sum += value.get();
			}
			context.write(key, new DoubleWritable(sum));
		}
	}
	public static void main(String[] args) throws
		Exception {Job job =
		Job.getInstance();
		job.setJobName("MapIntermediate");
		job.setJarByClass(Project1.class);
		MultipleInputs.addInputPath(job, new Path(args[0]),
TextInputFormat.class,MatriceMapperM.class);
		MultipleInputs.addInputPath(job, new Path(args[1]),
TextInputFormat.class,MatriceMapperN.class);
		job.setReducerClass(ReducerMxN.class);
		job.setMapOutputKeyClass(IntWritable.class);
		job.setMapOutputValueClass(Element.class);
		job.setOutputKeyClass(Pair.class);
		job.setOutputValueClass(DoubleWritable.class);
		job.setOutputFormatClass(TextOutputFormat.class);
		FileOutputFormat.setOutputPath(job, new
		Path(args[2]));job.waitForCompletion(true);
		Job job2 = Job.getInstance();
		job2.setJobName("MapFinalOutput");
		job2.setJarByClass(Project1.class);
		job2.setMapperClass(MapMxN.class);
		job2.setReducerClass(ReduceMxN.class);
		job2.setMapOutputKeyClass(Pair.class);
		job2.setMapOutputValueClass(DoubleWritable.class);
```

```
        job2.setOutputKeyClass(Pair.class);
        job2.setOutputValueClass(DoubleWritable.class);
        job2.setInputFormatClass(TextInputFormat.class);
        job2.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.setInputPaths(job2, new Path(args[2]));
        FileOutputFormat.setOutputPath(job2, new Path(args[3]));
        job2.waitForCompletion(true);
    }
}
```

Step 3. Compiling the program in particular folder named as operation

```
#!/bin/bash
rm -rf multiply.jar classes
module  load
hadoop/2.6.0
mkdir -p
classes
javac -d classes -cp classes:`$HADOOP_HOME/bin/hadoop classpath`
Multiply.javajar cf multiply.jar -C classes .
echo "end"
```

Step 4. Running the program in particular folder named as operation

```
export
HADOOP_CONF_DIR=/home/$USER/cometcl
ustermodule load hadoop/2.6.0
myhadoop-
configure.sh
start-dfs.sh
start-yarn.sh
hdfs dfs -mkdir -p /user/$USER
hdfs dfs -put M-matrix-large.txt /user/$USER/M-
matrix-large.txthdfs dfs -put N-matrix-large.txt
/user/$USER/N-matrix-large.txt
hadoop jar multiply.jar edu.uta.cse6331.Multiply /user/$USER/M-matrix-
large.txt
/user/$USER/N-matrix-large.txt /user/$USER/intermediate
/user/$USER/outputrm -rf output-distr
mkdir output-distr
hdfs dfs -get /user/$USER/output/part* output-distr
```

Output:

```
module  load
hadoop/2.6.0
rm -rf output
intermediate
hadoop --config $HOME jar multiply.jar edu.uta.cse6331.Multiply M-matrix-small.txt N-
matrix-small.txt intermediate output
```

stop-yarn.sh
stop-dfs.sh
myhadoop-cleanup.sh

RESULT

Thus, the program to Implement of Matrix Multiplication with Hadoop Map Reduce cluster was written, executed and verified successfully.

Ex.No:4	IMPLEMENTATION OF WORD COUNT PROGRAMS USING MAP REDUCE
Date:	

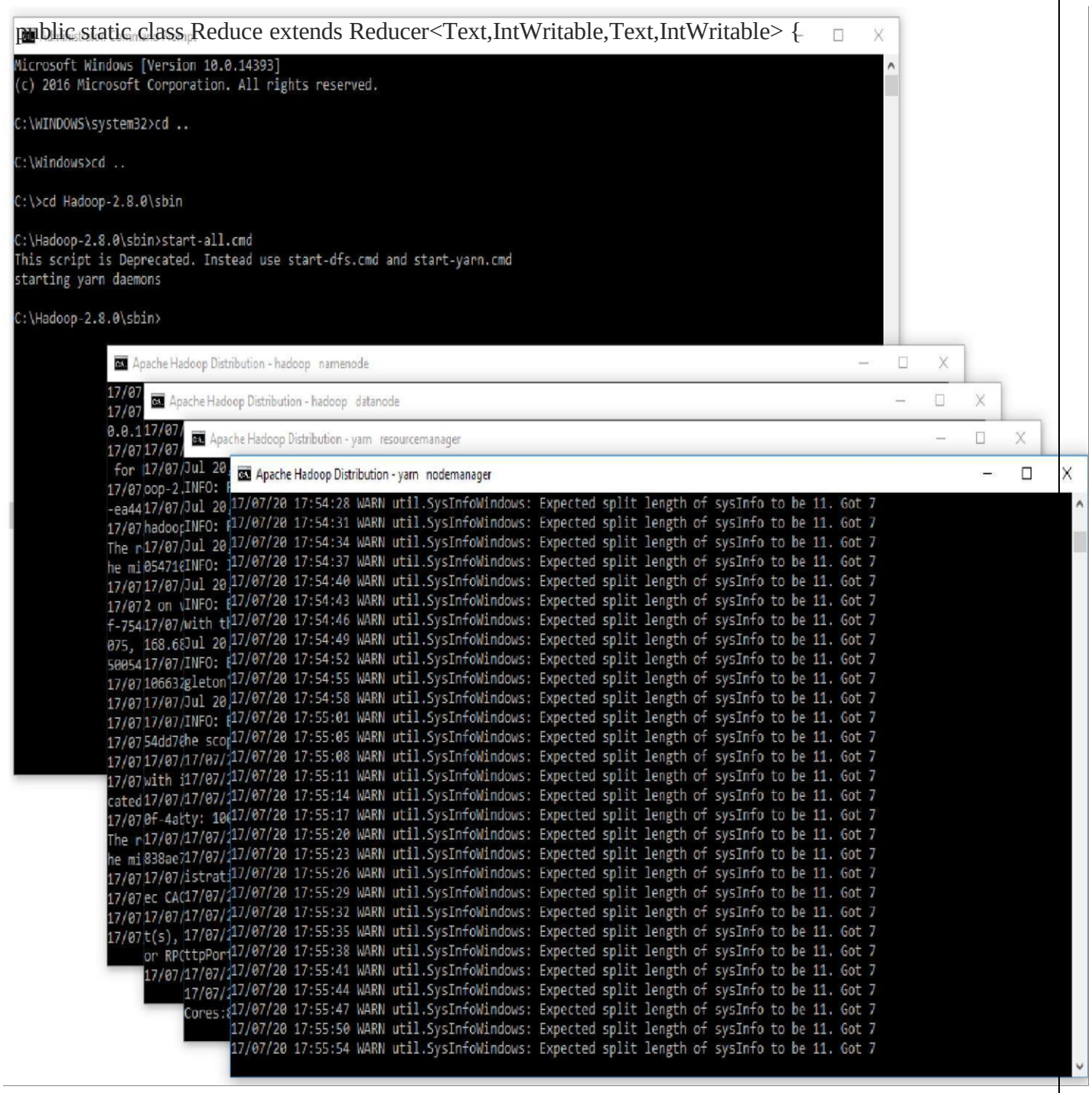
AIM:

To write a program to implement MapReduce application for word counting on Hadoop cluster

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);
}
}
```

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

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

We will understand the code for each of these three parts sequentially.

Mapper code:
public static class Map extends

1. Create an input directory in HDFS.

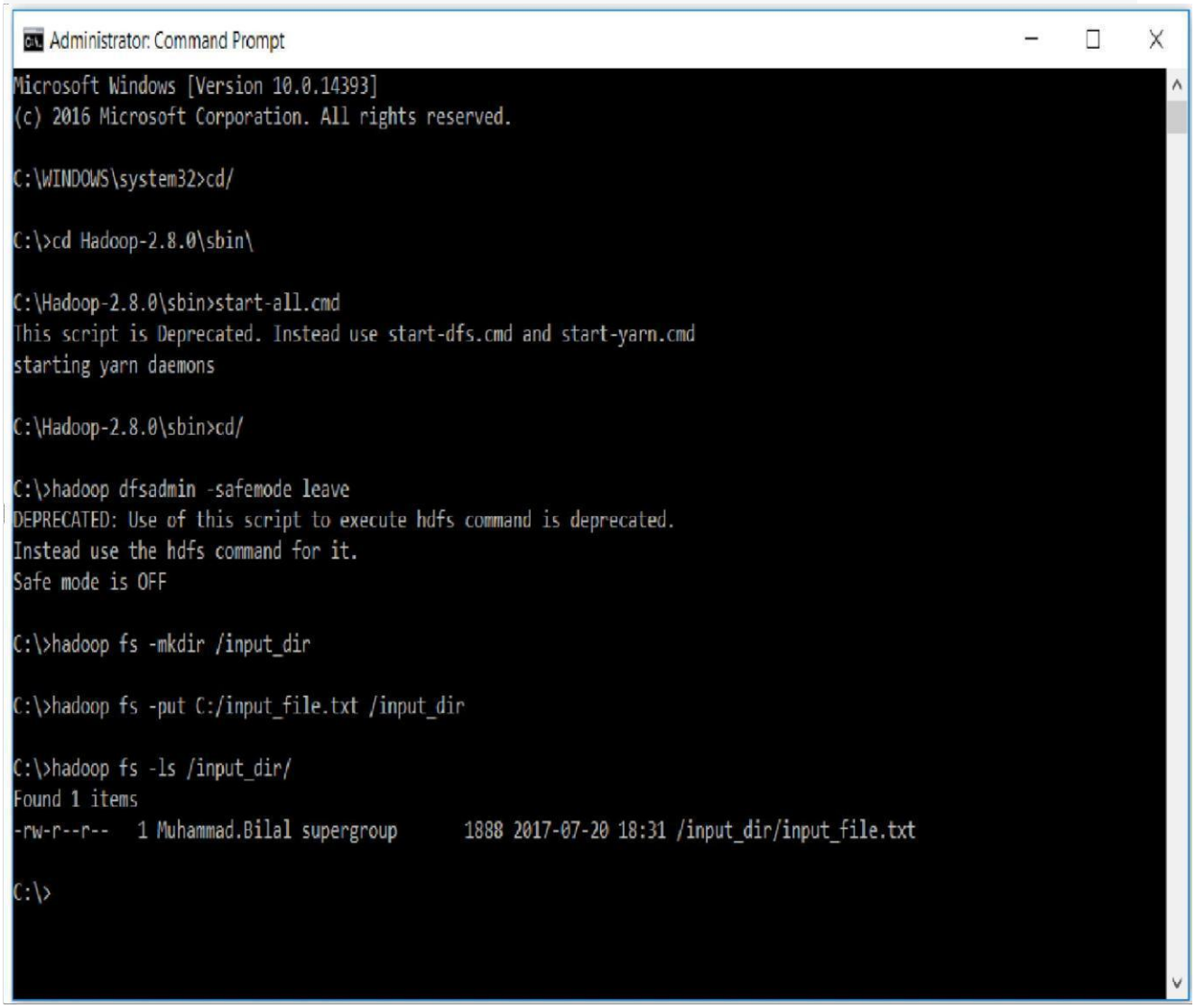
hadoop fs -mkdir /input_dir

2. Copy the input text file named input_file.txt in the input directory (input_dir)of HDFS.

hadoop fs -put C:/input_file.txt /input_dir

3. Verify input_file.txt available in HDFS input directory (input_dir)

hadoop fs -ls /input_dir/



```
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));
}
```

• We have created a class Map that extends the classMapper which is already defined in the MapReduce
Framework.
• We define the data types of input and output key/valuepair after the class declaration using angle brackets.
• Both the input and output of the Mapper is a key/valuepair.
• Input:
◦ The key is nothing but the offset of each line in
the text file:LongWritable
◦ The value is each individual line (as shown in the
figure at the right): 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 tokenizedeach word and assigned them a hardcoded value equal
to 1.

Reducer Code:

```
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));
```

1. Verify content of the copied file.

```
C:\>hadoop fs -ls /input_dir/
Found 1 items
-rw-r--r--   1 Muhammad.Bilal supergroup       1888 2017-07-20 18:31 /input_dir/input_file.txt

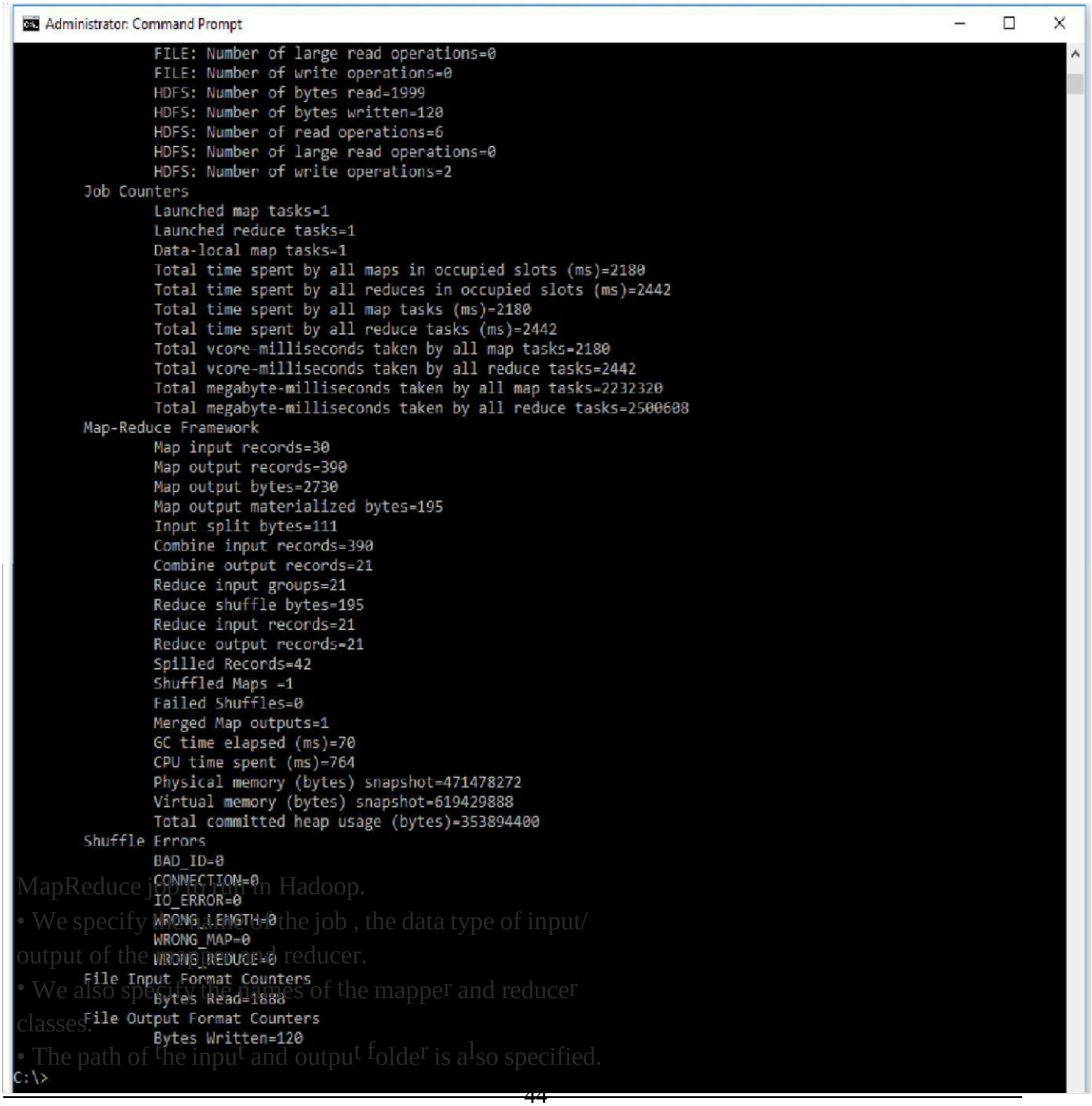
C:\>hadoop dfs -cat /input_dir/input_file.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
23   23   27  43   24   25   26   26   26   26   25   26  25
26   27   28  28   28   30   31   31   31   30   30   30  29
31   32   32  32   33   34   35   36   36   34   34   34  34
39   38   39  39   39   41   42   43   40   39   38   38  40
38   39   39  39   39   41   41   41   28   40   39   39  45
23   23   27  43   24   25   26   26   26   26   25   26  25
26   27   28  28   28   30   31   31   31   30   30   30  29
31   32   32  32   33   34   35   36   36   34   34   34  34
39   38   39  39   39   41   42   43   40   39   38   38  40
38   39   39  39   39   41   41   41   28   40   39   39  45
23   23   27  43   24   25   26   26   26   26   25   26  25
26   27   28  28   28   30   31   31   31   30   30   30  29
31   32   32  32   33   34   35   36   36   34   34   34  34
39   38   39  39   39   41   42   43   40   39   38   38  40
38   39   39  39   39   41   41   41   28   40   39   39  45
23   23   27  43   24   25   26   26   26   26   25   26  25
26   27   28  28   28   30   31   31   31   30   30   30  29
31   32   32  32   33   34   35   36   36   34   34   34  34
39   38   39  39   39   41   42   43   40   39   38   38  40
38   39   39  39   39   41   41   41   28   40   39   39  45
23   23   27  43   24   25   26   26   26   26   25   26  25
26   27   28  28   28   30   31   31   31   30   30   30  29
31   32   32  32   33   34   35   36   36   34   34   34  34
39   38   39  39   39   41   42   43   40   39   38   38  40
38   39   39  39   39   41   41   41   28   40   39   39  45
```

}
}
• We have created a class Reduce which extends class
Reducer like that of Mapper.
• We define the data types of input and output key/valuepair after the class declaration using angle brackets as
done for Mapper.
Both the input and the output of the Reducer is a keyvalue pair.
• Input:
◦ The key nothing but those unique words whichhave 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 thelist corresponding to each key and produced the final
answer.
• In general, a single reducer is created for each of theunique words, but, you can specify the number of
reducer in mapred-site.xml.
Driver Code:
```
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]));
```

• In the driver class, we set the configuration of our

1. Run MapReduceClient.jar and also provide input and out directories.

hadoop jar C:/MapReduceClient.jar wordcount /input_dir /output_dir



MapReduce job flow 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.

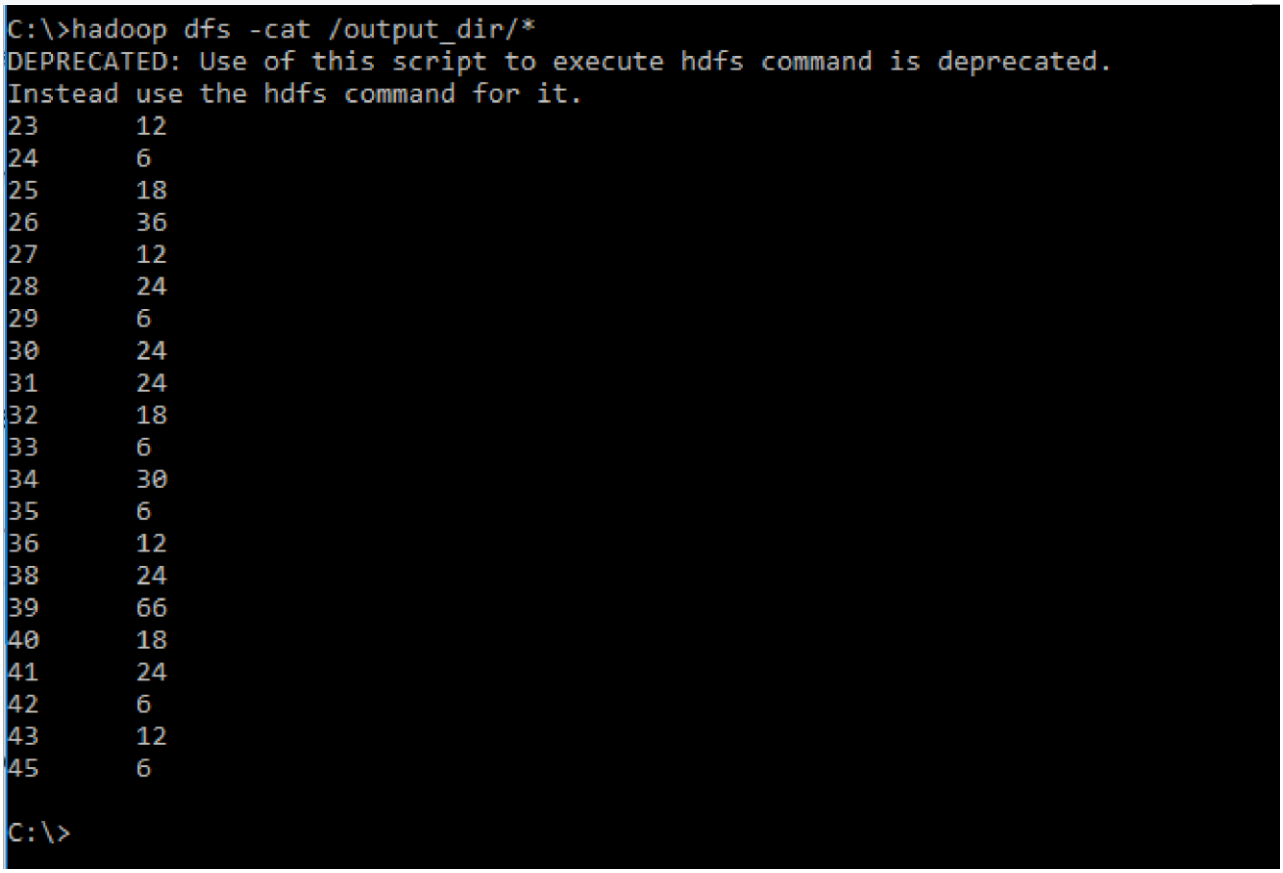
Run the MapReduce code:

The command for running a MapReduce code is:

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

1. Verifycontent for generated output file.

hadoop dfs -cat /output_dir/*



Some Other useful commands

3) To leave Safe mode

hadoop dfsadmin –safemode leave

4) To delete file from HDFS directory

hadoop fs -rm -r /iutput_dir/input_file.txt

5) To delete directory from HDFS directory

hadoop fs -rm -r /iutput_dir

RESULT

Thus, the java program to implement Mapreduce application for word counting on hadoop cluster was written, executed and verified successfully.

Ex.No:5	INSTALLATION OF HIVE ALONG WITH PRACTICE EXAMPLES
Date:	

AIM:

To write a program to Installation of Hive along with practice examples

PROCEDURE:

Step 1: Verifying JAVA Installation

Java must be installed on your system before installing Hive. Let us verify java installation using the following command:

```
$ java –version
```

If Java is already installed on your system, you get to see the following response:

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

If java is not installed in your system, then follow the steps given below for installing java.

Installing Java

Step I:

Download java (JDK <latest version> - X64.tar.gz) by visiting the following link http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html.

Then jdk-7u71-linux-x64.tar.gz will be downloaded onto your system.

Step II:

Generally you will find the downloaded java file in the Downloads folder. Verify it and extract the jdk-7u71-linux-x64.gz file using the following commands.

```
$ cd Downloads/
$ ls
jdk-7u71-linux-x64.gz
$ tar zxf jdk-7u71-linux-x64.gz
$ ls
jdk1.7.0_71 jdk-7u71-linux-x64.gz
```

Step III:

To make java available to all the users, you have to move it to the location "/usr/local/". Open root, and type the following commands.

```
$ su
password:
# mv jdk1.7.0_71 /usr/local/
# exit
```

Step IV:

For setting up PATH and JAVA_HOME variables, add the following commands to ~/.bashrc file.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
export PATH=$PATH:$JAVA_HOME/bin
```

Now apply all the changes into the current running system.

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

Step V:

Use the following commands to configure java alternatives:

```
# alternatives --install /usr/bin/java/java/usr/local/java/bin/java 2

# alternatives --install /usr/bin/javac/javac/usr/local/java/bin/javac 2

# alternatives --install /usr/bin/jar/jar/usr/local/java/bin/jar 2

# alternatives --set java/usr/local/java/bin/java

# alternatives --set javac/usr/local/java/bin/javac

# alternatives --set jar/usr/local/java/bin/jar
```

Now verify the installation using the command java -version from the terminal as explained above.

Step 2: Verifying Hadoop Installation

Hadoop must be installed on your system before installing Hive. Let us verify the Hadoop installation using the following command:

```
$ hadoop version
```

If Hadoop is already installed on your system, then you will get the following response:

```
Hadoop 2.4.1 Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiled by hortonmu on 2013-10-07T06:28Z
Compiled with protoc 2.5.0
From source with checksum 79e53ce7994d1628b240f09af91e1af4
```

If Hadoop is not installed on your system, then proceed with the following steps:

Downloading Hadoop

Download and extract Hadoop 2.4.1 from Apache Software Foundation using the following commands.

```
$ su
password:
# cd /usr/local
# wget http://apache.claz.org/hadoop/common/hadoop-2.4.1/
hadoop-2.4.1.tar.gz
# tar xzf hadoop-2.4.1.tar.gz
# mv hadoop-2.4.1/* to hadoop/
# exit
```

Installing Hadoop in Pseudo Distributed Mode

The following steps are used to install Hadoop 2.4.1 in pseudo distributed mode.

Step I: Setting up Hadoop

You can set Hadoop environment variables by appending the following commands to **~/.bashrc** file.

```
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native export
PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

Now apply all the changes into the current running system.

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

Step II: Hadoop Configuration

You can find all the Hadoop configuration files in the location "$HADOOP_HOME/etc/hadoop". You need to make suitable changes in those configuration files according to your Hadoop infrastructure.

```
$ cd $HADOOP_HOME/etc/hadoop
```

In order to develop Hadoop programs using java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing **JAVA_HOME** value with the location of java in your system.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
```

Given below are the list of files that you have to edit to configure Hadoop.

core-site.xml

The **core-site.xml** file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and the size of Read/Write buffers.

Open the core-site.xml and add the following properties in between the <configuration> and </configuration> tags.

```
<configuration>
  <property>
```

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

</configuration>
```

hdfs-site.xml

The **hdfs-site.xml** file contains information such as the value of replication data, the namenode path, and the datanode path of your local file systems. It means the place where you want to store the Hadoop infra.

Let us assume the following data.

```
dfs.replication (data replication value) = 1

(In the following path /hadoop/ is the user name.
hadoopinfra/hdfs/namenode is the directory created by hdfs file system.)

namenode path = //home/hadoop/hadoopinfra/hdfs/namenode

(hadoopinfra/hdfs/datanode is the directory created by hdfs file system.)
datanode path = //home/hadoop/hadoopinfra/hdfs/datanode
```

Open this file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>

  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value >
  </property>

</configuration>
```

Note: In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.

yarn-site.xml

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>

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

</configuration>
```

mapred-site.xml

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, you need to copy the file from mapred-site,xml.template to mapred-site.xml file using the following command.

```
$ cp mapred-site.xml.template mapred-site.xml
```

Open **mapred-site.xml** file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>

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

</configuration>
```

Verifying Hadoop Installation

The following steps are used to verify the Hadoop installation.

Step I: Name Node Setup

Set up the namenode using the command “hdfs namenode -format” as follows.

```
$ cd ~
$ hdfs namenode -format
```

The expected result is as follows.

```
10/24/14 21:30:55 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = localhost/192.168.1.11
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.4.1
...
...
```

```
10/24/14 21:30:56 INFO common.Storage: Storage directory
/home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.
10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to
retain 1 images with txid >= 0
10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0
10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at localhost/192.168.1.11
 ************************************************************/
```

Step II: Verifying Hadoop dfs

The following command is used to start dfs. Executing this command will start your Hadoop file system.

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

The expected output is as follows:

```
10/24/14 21:37:56
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hadoop/hadoop-2.4.1/logs/hadoop-hadoop-namenode-
localhost.out
localhost: starting datanode, logging to /home/hadoop/hadoop-2.4.1/logs/hadoop-hadoop-datanode-
localhost.out
Starting secondary namenodes [0.0.0.0]
```

Step III: Verifying Yarn Script

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

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

The expected output is as follows:

```
starting yarn daemons
starting resourcemanager, logging to /home/hadoop/hadoop-2.4.1/logs/yarn-hadoop-resourcemanager-
localhost.out
localhost: starting nodemanager, logging to /home/hadoop/hadoop-2.4.1/logs/yarn-hadoop-
nodemanager-localhost.out
```

Step IV: Accessing Hadoop on Browser

The default port number to access Hadoop is 50070. Use the following url to get Hadoop services on your browser.

```
http://localhost:50070/
```

Step V: Verify all applications for cluster

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

```

```

Step 3: Downloading Hive

We use hive-0.14.0 in this tutorial. You can download it by visiting the following link http://apache.petsads.us/hive/hive-0.14.0/. Let us assume it gets downloaded onto the /Downloads directory. Here, we download Hive archive named "apache-hive-0.14.0-bin.tar.gz" for this tutorial. The following command is used to verify the download:

```
$ cd Downloads
$ ls
```

On successful download, you get to see the following response:

```
apache-hive-0.14.0-bin.tar.gz
```

Step 4: Installing Hive

The following steps are required for installing Hive on your system. Let us assume the Hive archive is downloaded onto the /Downloads directory.

Extracting and verifying Hive Archive

The following command is used to verify the download and extract the hive archive:

```
$ tar zxvf apache-hive-0.14.0-bin.tar.gz
$ ls
```

On successful download, you get to see the following response:

```
apache-hive-0.14.0-bin  apache-hive-0.14.0-bin.tar.gz
```

Copying files to /usr/local/hive directory

We need to copy the files from the super user "su -". The following commands are used to copy the files from the extracted directory to the /usr/local/hive" directory.

```
$ su -
passwd:

# cd /home/user/Download
# mv apache-hive-0.14.0-bin /usr/local/hive
# exit
```

Setting up environment for Hive

You can set up the Hive environment by appending the following lines to **~/.bashrc** file:

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
```

The following command is used to execute ~/.bashrc file.

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

Step 5: Configuring Hive

To configure Hive with Hadoop, you need to edit the **hive-env.sh** file, which is placed in the **$HIVE_HOME/conf** directory. The following commands redirect to Hive **config** folder and copy the template file:

```
$ cd $HIVE_HOME/conf
$ cp hive-env.sh.template hive-env.sh
```

Edit the **hive-env.sh** file by appending the following line:

```
export HADOOP_HOME=/usr/local/hadoop
```

Hive installation is completed successfully. Now you require an external database server to configure Metastore. We use Apache Derby database.

Step 6: Downloading and Installing Apache Derby

Follow the steps given below to download and install Apache Derby:

Downloading Apache Derby

The following command is used to download Apache Derby. It takes some time to download.

```
$ cd ~
$  wget  http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz
```

The following command is used to verify the download:

```
$ ls
```

On successful download, you get to see the following response:

```
db-derby-10.4.2.0-bin.tar.gz
```

Extracting and verifying Derby archive

The following commands are used for extracting and verifying the Derby archive:

```
$ tar zxvf db-derby-10.4.2.0-bin.tar.gz
$ ls
```

On successful download, you get to see the following response:

```
db-derby-10.4.2.0-bin  db-derby-10.4.2.0-bin.tar.gz
```

Copying files to /usr/local/derby directory

We need to copy from the super user "su -". The following commands are used to copy the files from the extracted directory to the /usr/local/derby directory:

```
$ su -
passwd:
# cd /home/user
```

```
# mv db-derby-10.4.2.0-bin /usr/local/derby
# exit
```

Setting up environment for DerbyYou can set up the Derby environment by appending the following lines to **~/.bashrc** file:

```
export DERBY_HOME=/usr/local/derby
export PATH=$PATH:$DERBY_HOME/bin
Apache Hive
18 export
CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar
```

The following command is used to execute **~/.bashrc** file:

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

Create a directory to store Metastore

Create a directory named data in $DERBY_HOME directory to store Metastore data.

```
$ mkdir $DERBY_HOME/data
```

Derby installation and environmental setup is now complete.

Step 7: Configuring Metastore of Hive

Configuring Metastore means specifying to Hive where the database is stored. You can do this by editing the hive-site.xml file, which is in the $HIVE_HOME/conf directory. First of all, copy the template file using the following command:

```
$ cd $HIVE_HOME/conf
$ cp hive-default.xml.template hive-site.xml
```

Edit **hive-site.xml** and append the following lines between the <configuration> and </configuration> tags:

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:derby://localhost:1527/metastore_db;create=true  </value>
  <description>JDBC connect string for a JDBC metastore </description>
</property>
```

Create a file named jpox.properties and add the following lines into it:

```
javax.jdo.PersistenceManagerFactoryClass =

org.jpox.PersistenceManagerFactoryImpl
org.jpox.autoCreateSchema = false
org.jpox.validateTables = false
org.jpox.validateColumns = false
org.jpox.validateConstraints = false
org.jpox.storeManagerType = rdbms
```

```
org.jpox.autoCreateSchema = true
org.jpox.autoStartMechanismMode = checked
org.jpox.transactionIsolation = read_committed
javax.jdo.option.DetachAllOnCommit = true
javax.jdo.option.NontransactionalRead = true
javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver
javax.jdo.option.ConnectionURL = jdbc:derby://hadoop1:1527/metastore_db;create = true
javax.jdo.option.ConnectionUserName = APP
javax.jdo.option.ConnectionPassword = mine
```

Step 8: Verifying Hive Installation

Before running Hive, you need to create the **/tmp** folder and a separate Hive folder in HDFS. Here, we use the **/user/hive/warehouse** folder. You need to set write permission for these newly created folders as shown below:

```
chmod g+w
```

Now set them in HDFS before verifying Hive. Use the following commands:

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir /tmp
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse
```

The following commands are used to verify Hive installation:

```
$ cd $HIVE_HOME
$ bin/hive
```

On successful installation of Hive, you get to see the following response:

```
Logging initialized using configuration in jar:file:/home/hadoop/hive-0.9.0/lib/hive-common-
0.9.0.jar!/hive-log4j.properties
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201312121621_1494929084.txt
……................
hive>
```

The following sample command is executed to display all the tables:

```
hive> show tables;
OK
Time taken: 2.798 seconds
hive>
```

Hive Examples:

OBJECTIVE:

Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

PROGRAM :

SYNTAX for HIVE Database Operations
DATABASE Creation
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
Drop Database Statement
DROP DATABASE Statement DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];
Creating and Dropping Table in HIVE
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment] [ROW FORMAT row_format] [STORED AS file_format]
Loading Data into table log_data
Syntax:
LOAD DATA LOCAL INPATH '<path>/u.data' OVERWRITE INTO TABLE u_data;
Alter Table in HIVE
Syntax
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
Creating and Dropping View
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...)] [COMMENT table_comment] AS SELECT ...
Dropping View
Syntax:
DROP VIEW view_name
Functions in HIVE
String Functions:- round(), ceil(), substr(), upper(), reg_exp() etc
Date and Time Functions:- year(), month(), day(), to_date() etc
Aggregate Functions :- sum(), min(), max(), count(), avg() etc
43
INDEXES
CREATE INDEX index_name ON TABLE base_table_name (col_name, ...)
AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property_name=property_value, ...)]
[IN TABLE index_table_name]

[PARTITIONED BY (col_name, ...)]
[
[ROW FORMAT ...] STORED AS ...
| STORED BY ...
]
[LOCATION hdfs_path]
[TBLPROPERTIES (...)]
Creating Index
CREATE INDEX index_ip ON TABLE log_data(ip_address) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED
REBUILD;
Altering and Inserting Index
ALTER INDEX index_ip_address ON log_data REBUILD;
Storing Index Data in Metastore
SET
hive.index.compact.file=/home/administrator/Desktop/big/metastore_db/tmp/index_ipadd
ress_result;
SET
hive.input.format=org.apache.hadoop.hive.ql.index.compact.HiveCompactIndexInputFor
mat;
Dropping Index
DROP INDEX INDEX_NAME on TABLE_NAME;

RESULT:

Thus, the program to Installation of Hive along with practice examples was written, executed and verified successfully.

Ex.No:6	INSTALLATION OF HBASE, INSTALLING THRIFT ALONG WITH EXAMPLES
Date:	

AIM:

To write a procedure for Installation of HBase, Installing thrift along with examples.

PROCEDURE:

Installing HBase

We can install HBase in any of the three modes: Standalone mode, Pseudo Distributed mode, and Fully Distributed mode.

Installing HBase in Standalone Mode

Download the latest stable version of HBase form http://www.interior- dsgn.com/apache/hbase/stable/ using "wget" command, and extract it using the tar "zxvf" command. See the following command.

```
$cd usr/local/
$wget http://www.interior-dsgn.com/apache/hbase/stable/hbase-0.98.8-
hadoop2-bin.tar.gz
$tar -zxvf hbase-0.98.8-hadoop2-bin.tar.gz
```

Shift to super user mode and move the HBase folder to /usr/local as shown below.

```
$su
$password: enter your password here
mv hbase-0.99.1/* Hbase/
```

Configuring HBase in Standalone Mode

Before proceeding with HBase, you have to edit the following files and configure HBase.

hbase-env.sh

Set the java Home for HBase and open **hbase-env.sh** file from the conf folder. Edit JAVA_HOME environment variable and change the existing path to your current JAVA_HOME variable as shown below.

```
cd /usr/local/Hbase/conf
gedit hbase-env.sh
```

This will open the env.sh file of HBase. Now replace the existing **JAVA_HOME** value with your current value as shown below.

export JAVA_HOME=/usr/lib/jvm/java-1.7.0

hbase-site.xml

This is the main configuration file of HBase. Set the data directory to an appropriate location by opening the HBase home folder in /usr/local/HBase. Inside the conf folder, you will find several files, open the **hbase-site.xml** file as shown below.

#cd /usr/local/HBase/
#cd conf
gedit hbase-site.xml

Inside the **hbase-site.xml** file, you will find the <configuration> and </configuration> tags. Within them, set the HBase directory under the property key with the name "hbase.rootdir" as shown below.

```
<configuration>
  //Here you have to set the path where you want HBase to store its files.
  <property>
    <name>hbase.rootdir</name>
    <value>file:/home/hadoop/HBase/HFiles</value>
  </property>

  //Here you have to set the path where you want HBase to store its built in zookeeper  files.
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/hadoop/zookeeper</value>
  </property>
</configuration>
```

With this, the HBase installation and configuration part is successfully complete. We can start HBase by using **start-hbase.sh** script provided in the bin folder of HBase. For that, open HBase Home Folder and run HBase start script as shown below.

$cd /usr/local/HBase/bin
$./start-hbase.sh

If everything goes well, when you try to run HBase start script, it will prompt you a message saying that HBase has started.

starting master, logging to /usr/local/HBase/bin/../logs/hbase-tpmaster-localhost.localdomain.out

Installing HBase in Pseudo-Distributed Mode

Let us now check how HBase is installed in pseudo-distributed mode.

Configuring HBase

Before proceeding with HBase, configure Hadoop and HDFS on your local system or on a remote system and make sure they are running. Stop HBase if it is running.

hbase-site.xml

Edit hbase-site.xml file to add the following properties.

```
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
```

It will mention in which mode HBase should be run. In the same file from the local file system, change the hbase.rootdir, your HDFS instance address, using the hdfs:// URI syntax. We are running HDFS on the localhost at port 8030.

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:8030/hbase</value>
</property>
```

Starting HBase

After configuration is over, browse to HBase home folder and start HBase using the following command.

```
$cd /usr/local/HBase
$bin/start-hbase.sh
```

Note: Before starting HBase, make sure Hadoop is running.

Checking the HBase Directory in HDFS

HBase creates its directory in HDFS. To see the created directory, browse to Hadoop bin and type the following command.

```
$ ./bin/hadoop fs -ls /hbase
```

If everything goes well, it will give you the following output.

```
Found 7 items
drwxr-xr-x - hbase users 0 2014-06-25 18:58 /hbase/.tmp
drwxr-xr-x - hbase users 0 2014-06-25 21:49 /hbase/WALs
drwxr-xr-x - hbase users 0 2014-06-25 18:48 /hbase/corrupt
drwxr-xr-x - hbase users 0 2014-06-25 18:58 /hbase/data
-rw-r--r-- 3 hbase users 42 2014-06-25 18:41 /hbase/hbase.id
-rw-r--r-- 3 hbase users 7 2014-06-25 18:41 /hbase/hbase.version
drwxr-xr-x - hbase users 0 2014-06-25 21:49 /hbase/oldWALs
```

Starting and Stopping a Master

Using the "local-master-backup.sh" you can start up to 10 servers. Open the home folder of HBase, master and execute the following command to start it.

```
$ ./bin/local-master-backup.sh 2 4
```

To kill a backup master, you need its process id, which will be stored in a file named **"/tmp/hbase-USER-X-master.pid."** you can kill the backup master using the following command.

```
$ cat /tmp/hbase-user-1-master.pid |xargs kill -9
```

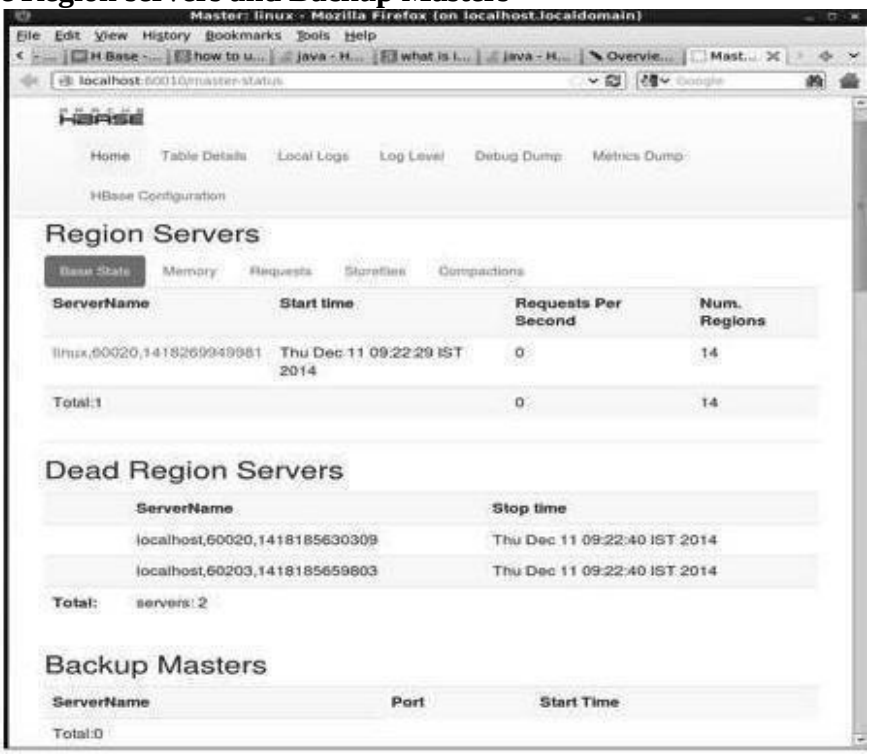
HBase Web Interface

To access the web interface of HBase, type the following url in the browser.

http://localhost:60010

This interface lists your currently running Region servers, backup masters and HBase tables.

HBase Region servers and Backup Masters



Starting and Stopping RegionServers

You can run multiple region servers from a single system using the following command.

$ .bin/local-regionservers.sh start 2 3

To stop a region server, use the following command.

$ .bin/local-regionservers.sh stop 3

Starting HBaseShell

After Installing HBase successfully, you can start HBase Shell. Below given are the sequence of steps that are to be followed to start the HBase shell. Open the terminal, and login as super user.

Start Hadoop File System

Browse through Hadoop home sbin folder and start Hadoop file system as shown below.

$cd $HADOOP_HOME/sbin
$start-all.sh

Start HBase

Browse through the HBase root directory bin folder and start HBase.

$cd /usr/local/HBase
$./bin/start-hbase.sh

Start HBase Master Server

This will be the same directory. Start it as shown below.

$./bin/local-master-backup.sh start 2 (number signifies specific
server.)

Start Region

Start the region server as shown below.

$./bin/./local-regionservers.sh start 3

Start HBase Shell

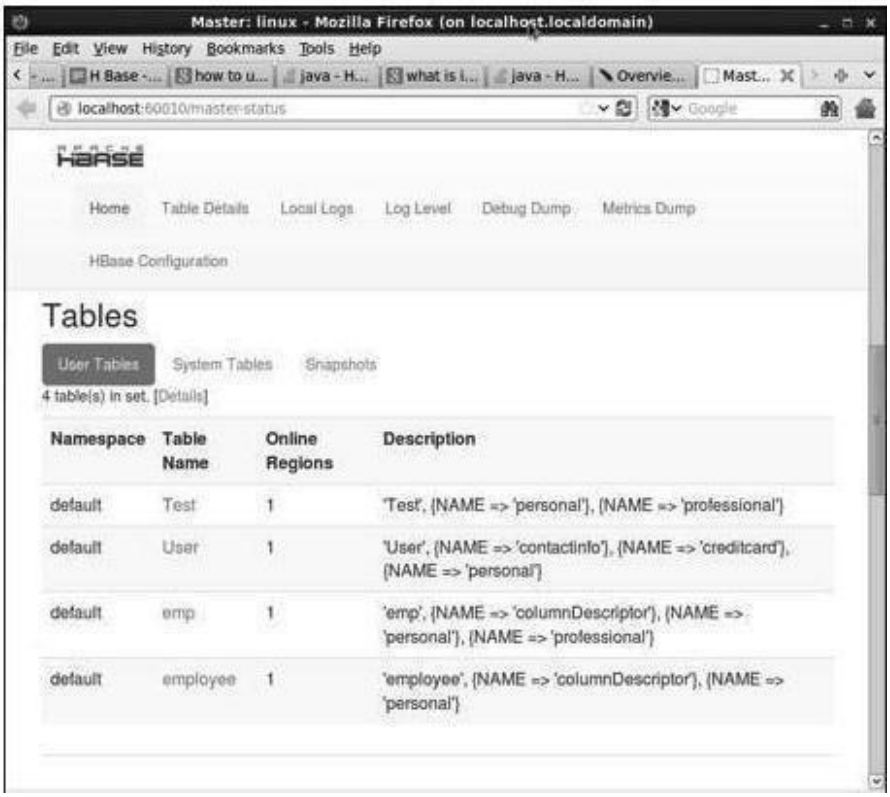
You can start HBase shell using the following command.

$cd bin
$./hbase shell

This will give you the HBase Shell Prompt as shown below.

2014-12-09 14:24:27,526 INFO [main] Configuration.deprecation:
hadoop.native.lib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.8-hadoop2, r6cfc8d064754251365e070a10a82eb169956d5fe, Fri
Nov 14 18:26:29 PST 2014
hbase(main):001:0>

HBase Tables



RESULT:

Thus, the program to Installation of HBase, Installing thrift along with examples was executed and verified successfully.

Ex.No:7	IMPORTING AND EXPORTING DATA FROM VARIOUS DATABASES
Date:	

AIM:

To write a procedure for Importing and exporting data from various databases

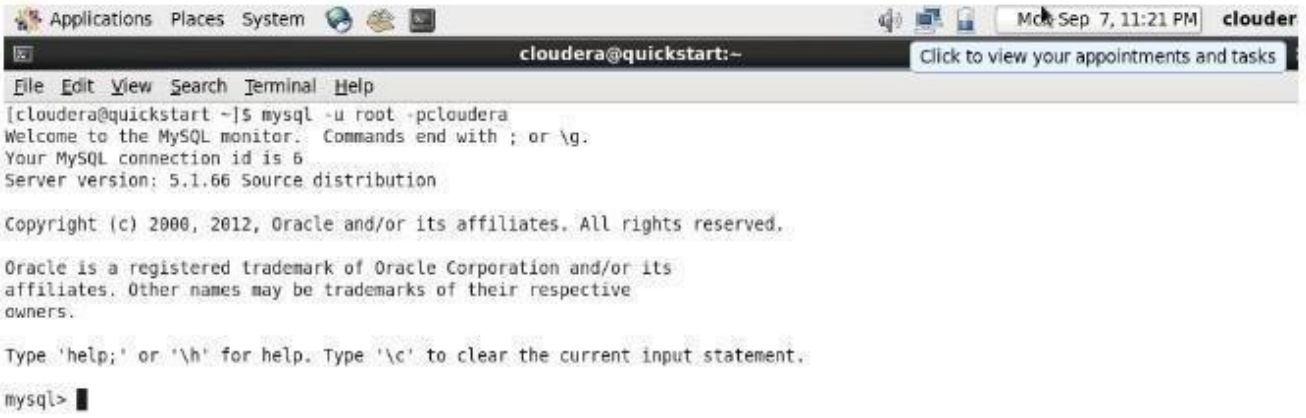
PROCEDURE:

SQOOP is basically used to transfer data from relational databases such as MySQL, Oracle to data warehouses such as Hadoop HDFS(Hadoop File System). Thus, when data is transferred from a relational database to HDFS, we say we are **importing data.** Otherwise, when we transfer data from HDFS to relational databases, we say we are **exporting data.**
Note: To import or export, the order of columns in both MySQL and Hive should be the same.

Importing data from MySQL to HDFS

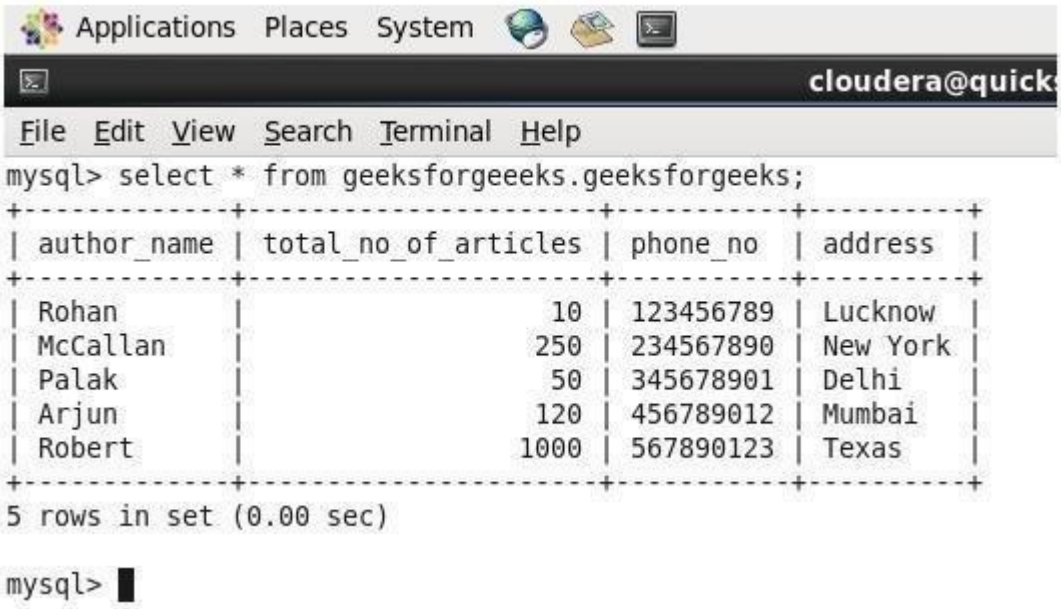
In order to store data into HDFS, we make use of Apache Hive which provides an SQL-like interface between the user and the Hadoop distributed file system (HDFS) which integrates Hadoop. We perform the following steps:

Step 1: Login into MySQL
mysql -u root –pcloudera



Step 2: Create a database and table and insert data.
create database geeksforgeeeks;
create table geeksforgeeeks.geeksforgeeks(author_name varchar(65), total_no_of_articles int,

phone_no int, address varchar(65));
insert into geeksforgeeks values(“Rohan”,10,123456789,”Lucknow”);



Step 3: Create a database and table in the hive where data should be imported.
create table geeks_hive_table(name string, total_articles int, phone_no int, address string) row format delimited fields terminated by „,“;

Step 4: Run below the import command on Hadoop.
sqoop import --connect \

jdbc:mysql://127.0.0.1:3306/database_name_in_mysql \

--username root --password cloudera \

--table table_name_in_mysql \

--hive-import --hive-table database_name_in_hive.table_name_in_hive \

--m 1

In the above code following things should be noted.

- **127.0.0.1** is localhost IP address.
- **3306** is the port number for MySQL.
- **m** is the number of mappers

Step 5: Check-in hive if data is imported successfully or not.

Exporting data from HDFS to MySQL

To export data into MySQL from HDFS, perform the following steps:

Step 1: Create a database and table in the hive.
create table hive_table_export(name string,company string, phone int, age int) row format delimited fields terminated by „„“;

Step 2: Insert data into the hive table.
insert into hive_table_export values("Ritik","Amazon",234567891,35);

Data in Hive table

Step 3: Create a database and table in MySQL in which data should be exported.

Step 4: Run the following command on Hadoop.

```
sqoop export --connect \
jdbc:mysql://127.0.0.1:3306/database_name_in_mysql \
 --table table_name_in_mysql \
 --username root --password cloudera \
 --export-dir /user/hive/warehouse/hive_database_name.db/table_name_in_hive \
 --m 1 \
 -- driver com.mysql.jdbc.Driver
 --input-fields-terminated-by ','
```

Step 5: Check-in MySQL if data is exported successfully or not.

RESULT:

Thus, the program to Importing and exporting data from various databases was, executed and verified successfully.