Practical 1

Aim:- Configure Hadoop cluster in pseudo distributed mode. Try Hadoop basic commands.

sbin/start-all.sh

To check the Hadoop services are up and running use the following command:

jps

rahul@rahul:~/hadoop-3.3.6-cdh5.3.2$ jps
2546 SecondaryNameNode
2404 DataNode
2295 NameNode
2760 ResourceManager
2874 NodeManager
4251 Jps

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

Syntax: bin/hdfs dfs -ls <path>

Example:
bin/hdfs dfs -ls /

It will print all the directories present in HDFS. bin directory contains executable so, bin/hdfs means we want the executable of hdfs particularly dfs(Distributed File System) commands.

mkdir: To create a directory. In Hadoop dfs there is no home directory by default. So let's first create it.

Syntax: bin/hdfs dfs -mkdir <folder name>

creating home directory:
hdfs/bin -mkdir /user
hdfs/bin -mkdir /user/username -> write the username of your computer

Example:
bin/hdfs dfs -mkdir /geeks => '/' means absolute path bin/hdfs dfs -mkdir geeks2 => Relative path -> the folder will be created relative to the home directory.

touchz: It creates an empty file.

Syntax: bin/hdfs dfs -touchz <file_path>

Example:
bin/hdfs dfs -touchz /geeks/myfile.txt

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

Syntax: bin/hdfs dfs -copyFromLocal <local file path> <dest(present on hdfs)>
Example:
Let's suppose we have a file AI.txt on Desktop which we want to copy to folder rahul present on hdfs.

bin/hdfs dfs -copyFromLocal ../Desktop/AI.txt /rahul
OR
bin/hdfs dfs -put ../Desktop/AI.txt /Rahul

cat: To print file contents.

Syntax: bin/hdfs dfs -cat <path>

Example:
// print the content of AI.txt present // inside rahul folder.
bin/hdfs dfs -cat /rahul/AI.txt ->

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

Syntax: bin/hdfs dfs -copyToLocal <<srcfile(on hdfs)> <local file dest>

Example:

bin/hdfs dfs -copyToLocal /rahul ../Desktop/hero
OR
bin/hdfs dfs -get /rahul/myfile.txt ../Desktop/hero myfile.txt from rahul folder will be copied to folder hero present on Desktop.

moveFromLocal: This command will move file from local to hdfs.

Syntax: bin/hdfs dfs -moveFromLocal <local src> <dest(on hdfs)>

Example:
bin/hdfs dfs -moveFromLocal ../Desktop/cutAndPaste.txt /Rahul

cp: This command is used to copy files within hdfs. Let's copy folder rahul to rahul_copied.

Syntax: bin/hdfs dfs -cp <src(on hdfs)> <dest(on hdfs)>

Example:
bin/hdfs -cp /rahul /rahul_copied

mv: This command is used to move files within hdfs. Let's cut-paste a file myfile.txt from rahul folder to geeks_copied.

Syntax: bin/hdfs dfs -mv <src(on hdfs)> <src(on hdfs)>

Example: bin/hdfs -mv /rahul/myfile.txt /rahul_copied

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

Syntax: bin/hdfs dfs -rmr <filename/directoryName>

Example: bin/hdfs dfs -rmr /rahul_copied -> It will delete all the content inside the directory then the directory itself.

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

Syntax: bin/hdfs dfs -du <dirName>

Example:
bin/hdfs dfs -du /Rahul

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

Syntax: bin/hdfs dfs -dus <dirName>

Example:
bin/hdfs dfs -dus /Rahul

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

Syntax: bin/hdfs dfs -stat <hdfs file>

Example:
bin/hdfs dfs -stat /Rahul

setrep: This command is used to change the replication factor of a file/directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs core-site.xml).
Example 1: To change the replication factor to 6 for rahul.txt stored in HDFS. bin/hdfs dfs -setrep -R -w 6 rahul.txt
Example 2: To change the replication factor to 4 for a directory rahulInput stored in HDFS. bin/hdfs dfs -setrep -R 4 /Rahul

Note: The **-w** means wait till the replication is completed. And **-R** means recursively, we use it for directories as they may also contain many files and folders inside them.

Practical 2

Aim:- Write Map Reduce code for following:

a. Count frequency of words from a large file.

MapReduce consists of 2 steps:
Map Function – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

Example – (Map function in Word Count)
Input
Set of data
Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN, BUS, buS, caR, CAR, car, BUS, TRAIN

Output
Convert into another set of data
(Key,Value)
(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Reduce Function – Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

Example – (Reduce function in Word Count)

Input
(output of Map function)

Set of Tuples
(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1),
(buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Output
Converts into smaller set of tuples
(BUS,7), (CAR,7), (TRAIN,4)
Work Flow of the Program
Workflow of MapReduce consists of 5 steps:

Fig. WorkFlow of MapReducing
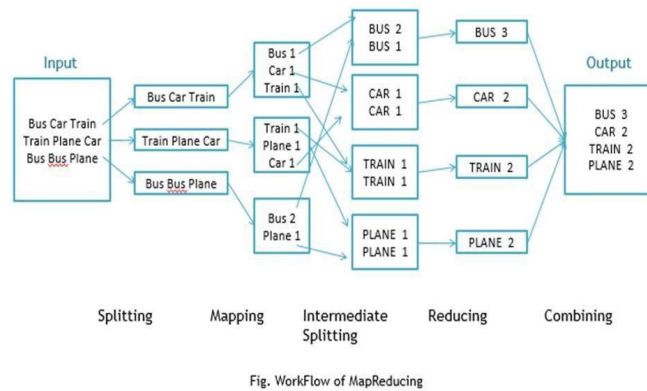
Splitting – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line ('\n').

Mapping – as explained above.

Intermediate splitting – the entire process in parallel on different clusters. In order to group them in
"Reduce Phase" the similar KEY data should be on the same cluster.

Reduce – it is nothing but mostly group by phase.

Combining – The last phase where all the data (individual result set from each cluster) is combined together to form a result.

Steps
1.   **Open Eclipse**> File > New > Java Project >(Name it – MRProgramsDemo) > Finish.

2.   **Right Click** > New > Package (Name it - PackageDemo) > Finish.

3.   **Right Click on Package** > New > Class (Name it - WordCount).

4.   **Add Following Reference Libraries**:

     a. Right Click on Project > Build Path> Add External

     i. /usr/lib/hadoop-0.20/hadoop-core.jar
     ii. U r/lib/hadoop-0.20/lib/Commons-cli-1.2.jar

5.   **Type the following code in java:**


```
package PackageDemo;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
```

```java
import org.apache.hadoop.io.Text;
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.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
public static void main(String [] args) throws Exception
{
Configuration c=new Configuration();
String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
Path input=new Path(files[0]);
Path output=new Path(files[1]);
Job j=new Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true)?0:1);
}
public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{
public void map(LongWritable key, Text value, Context con) throws IOException,
InterruptedException {
String line = value.toString();
String[] words=line.split(",");
for(String word: words )
{
Text outputKey = new Text(word.toUpperCase().trim());
IntWritable outputValue = new IntWritable(1);
con.write(outputKey, outputValue);
}

}
}
public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException,
InterruptedE xception {
int sum = 0;
for(IntWritable value : values)
{
sum += value.get();
}
con.write(word, new IntWritable(sum));
```

}
}
}

The above program consists of three classes:

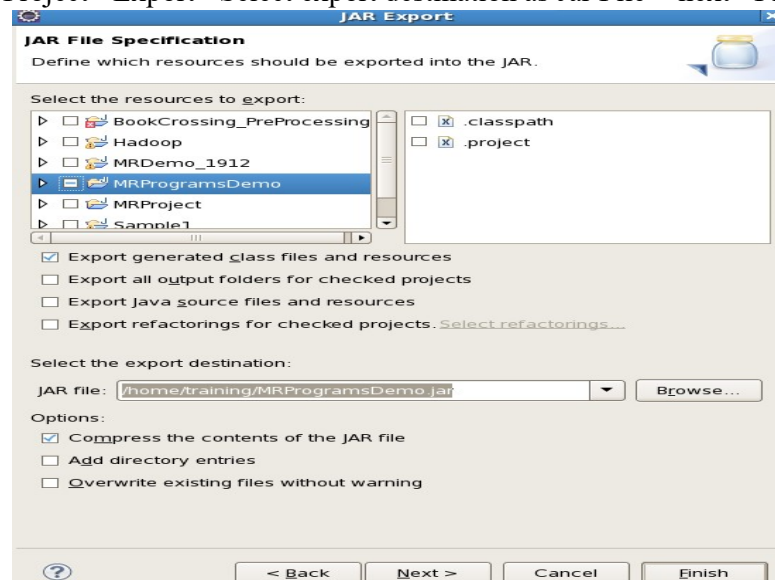Driver class (Public, void, static, or main; this is the entry point).
The Map class which extends the public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> and implements the Map function.
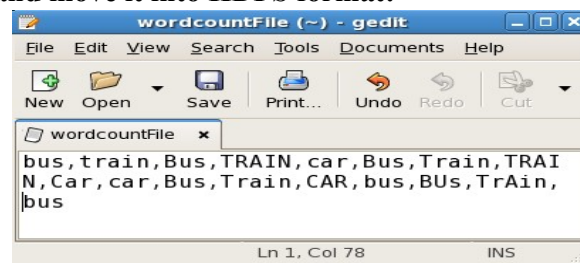The Reduce class which extends the public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> and implements the Reduce function.

6. Make a jar file

Right Click on Project> Export> Select export destination as Jar File > next> Finish.

7.   Take a text file and move it into HDFS format:

To move this into Hadoop directly, open the terminal and enter the following commands:

hadoop fs -put wordcountFile wordCountFile

8.   Run the jar file:

(Hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFile PathToOutputDirectry)

hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile MRDir1

9.   Open the result:
[rahul@localhost ~]$ **hadoop fs -ls MRDir1**

Found 3 items

-rw-r--r-- 1 training supergroup	0 2023-07-21 09:50 /user/rahul/MRDir1/_SUCCESS	
drwxr-xr-x - training supergroup	0 2023-07-21 09:50 /user/rahul/MRDir1/_logs	
-rw-r--r-- 1 training supergroup	20 2023-07-21 09:50 /user/rahul/MRDir1/part-r-00000	

[rahul@localhost ~]$ **hadoop fs -cat MRDir1/part-r-00000**
BUS 7
CAR 4
TRAIN 6

b. Find year wise maximum temperature using the weather data set which consists of year, month, and temperature.

Download Data set Step 1:
Index of /pub/data/uscrn/products/daily01/2020 (noaa.gov)
I have selected CRND0103-2020-AK_Fairbanks_11_NE.txt dataset for analysis of hot and cold days in Fairbanks, Alaska. We can get information about data from README.txt file available on the NCEI website.
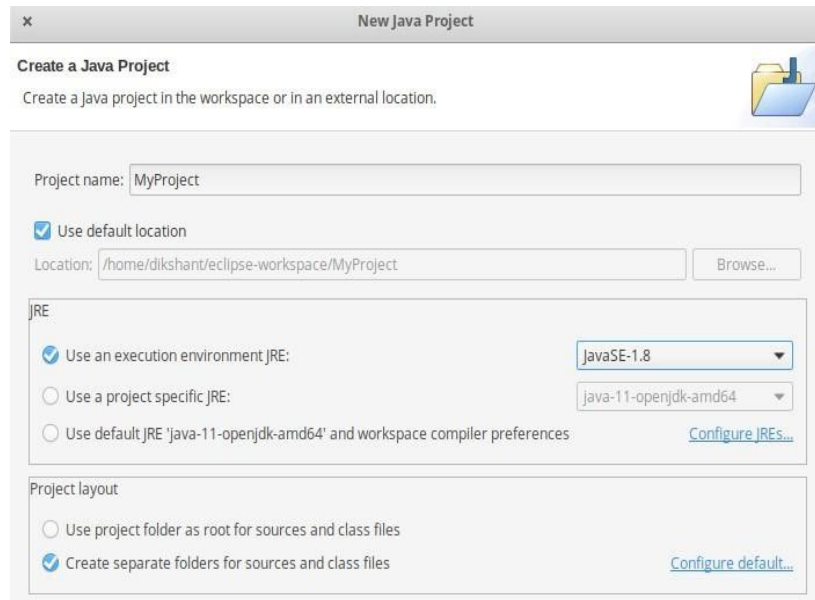
Step 2:
Below is the example of our dataset where column 6 and column 7 is showing Maximum and Minimum temperature, respectively.
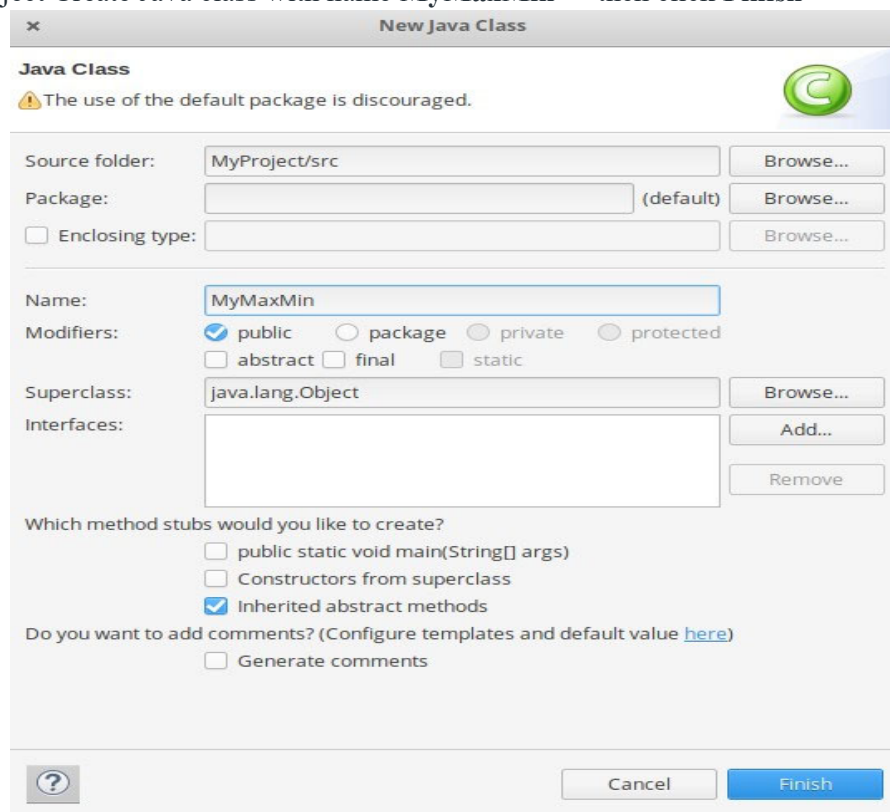

Step 3:

Make a project in Eclipse with below steps:
First Open **Eclipse** -> then select **File -> New -> Java Project** ->Name it **MyProject** -> then
select **use an execution environment** -> choose **JavaSE-1.8** then **next** -> **Finish**.

In this Project Create Java class with name **MyMaxMin** -> then click **Finish**

Copy the below source code to this **MyMaxMin** java class

MyMaxMin.java

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;

public class MyMaxMin {
```

// Mapper

```
/*MaxTemperatureMapper class is static and extends Mapper abstract class having four Hadoop
generics type LongWritable, Text, Text, Text. */


public static class MaxTemperatureMapper extends
Mapper<LongWritable, Text, Text, Text> {

/**
* @method map
* This method takes the input as a text data type.
* Now leaving the first five tokens, it takes
* 6th token is taken as temp_max and
* 7th token is taken as temp_min. Now * temp_max > 30 and temp_min < 15 are * passed to the
  reducer.
*/

// the data in our data set with
// this value is inconsistent data public static final int MISSING = 9999;

@Override public void map(LongWritable arg0, Text Value, Context context) throws
IOException, InterruptedException {

// Convert the single row(Record) to
// String and store it in String
// variable name line

String line = Value.toString();
```

10

```
// Check for the empty line
if (!(line.length() == 0)) {

// from character 6 to 14 we have
// the date in our dataset
String date = line.substring(6, 14);

// similarly we have taken the maximum
// temperature from 39 to 45 characters
float temp_Max = Float.parseFloat(line.substring(39, 45).trim());

// similarly we have taken the minimum
// temperature from 47 to 53 characters

float temp_Min = Float.parseFloat(line.substring(47, 53).trim());
// if maximum temperature is
// greater than 30, it is a hot day if (temp_Max > 30.0) {
```

// Hot day
```
        context.write(new    Text("The    Day    is    Hot    Day    :"    +    date),    new
Text(String.valueOf(temp_Max)));
        }
// if the minimum temperature is // less than 15, it is a cold day if (temp_Min < 15) {
```

// Cold day context.write(new Text("The Day is Cold Day :" + date),
```
new Text(String.valueOf(temp_Min)));
}
}
}


}
```

// Reducer

```
        /*MaxTemperatureReducer class is static and extends Reducer abstract class having four
Hadoop generics type Text, Text, Text, Text. */
 public static class MaxTemperatureReducer extends Reducer<Text, Text, Text, Text> {
 /**
* @method reduce
* This method takes the input as key and
* list of values pair from the mapper, * it does aggregation based on keys and * produces the final
  context.
  */

public void reduce(Text Key, Iterator<Text> Values, Context context)throws IOException,
InterruptedException {
```

```
                // putting all the values in
                // temperature variable of type String
                        String temperature = Values.next().toString();
                        context.write(Key, new Text(temperature)); }


        }
        /*
    * @method main
    * This method is used for setting * all the configuration properties. * It acts as a driver for map-
      reduce * code.
        */
public static void main(String[] args) throws Exception {
                // reads the default configuration of the
                // cluster from the configuration XML files
                Configuration conf = new Configuration();

                // Initializing the job with the
                // default configuration of the cluster
                Job job = new Job(conf, "weather example");

                // Assigning the driver class name
                job.setJarByClass(MyMaxMin.class);
                // Key type coming out of mapper
                job.setMapOutputKeyClass(Text.class);

                // value type coming out of mapper
                job.setMapOutputValueClass(Text.class);

                // Defining the mapper class name
                job.setMapperClass(MaxTemperatureMapper.class);

                // Defining the reducer class name
                job.setReducerClass(MaxTemperatureReducer.class);

                // Defining input Format class which is responsible to parse the dataset into a key value pair
                job.setInputFormatClass(TextInputFormat.class);

                // Defining output Format class which is responsible to parse the dataset into a key value
                pair job.setOutputFormatClass(TextOutputFormat.class);

                // setting the second argument as a path in a path variable
                Path OutputPath = new Path(args[1]);

                // Configuring the input path from the filesystem into the job
                FileInputFormat.addInputPath(job, new Path(args[0]));

                // Configuring the output path from the filesystem into the job
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

// deleting the context 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);
}
}

Now we need to add external jar for the packages that we have import. Download the jar package Hadoop Common and Hadoop MapReduce Core according to your Hadoop version. You can check Hadoop Version:

hadoop version

```
dikshant@dikshant-Inspiron-5567:~$ hadoop version
Hadoop 2.9.0
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r 756ebc8394e473ac25feac05fa493f6d612e6c50
Compiled by arsuresh on 2017-11-13T23:15Z
Compiled with protoc 2.5.0
From source with checksum 0a76a9a32a5257331741f8d5932f183
```

Now we add these external jars to our **MyProject.** Right Click on **MyProject** -> then select **Build Path**-> Click on **Configure Build Path** and select **Add External jars**…. and add jars from it's download location then click -> **Apply and Close.**
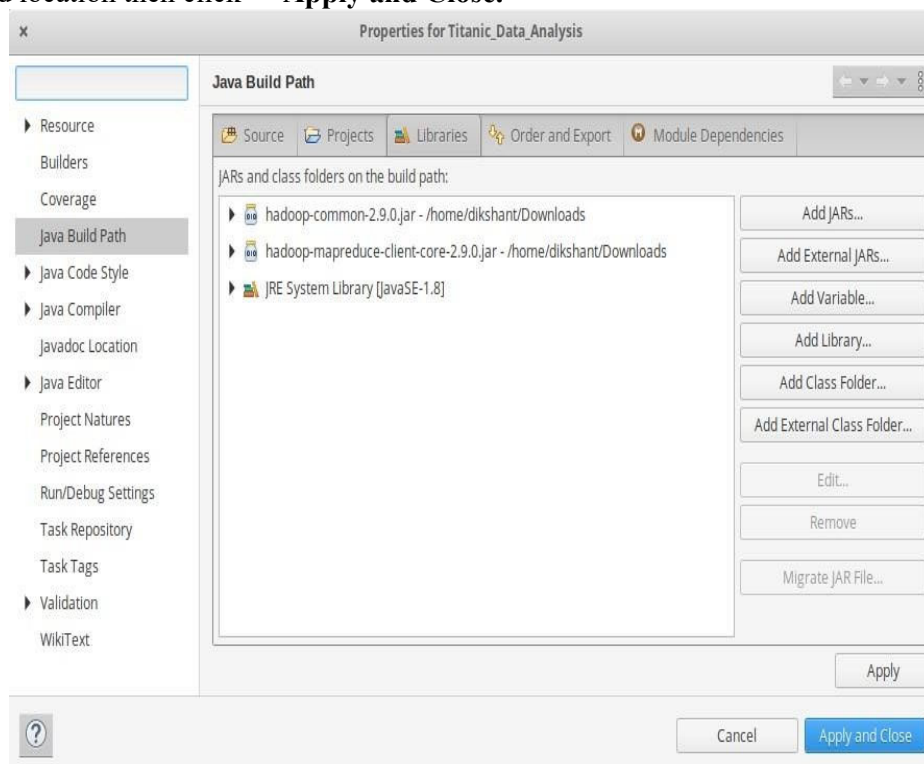
Now export the project as jar file. Right-click on **MyProject** choose **Export**. and go to **Java** ->
JAR file click -> Next and choose your export destination then **click -> Next**. choose Main
Class as **MyMaxMin** by
clicking -> Browse and then **click -> Finish -> Ok.**
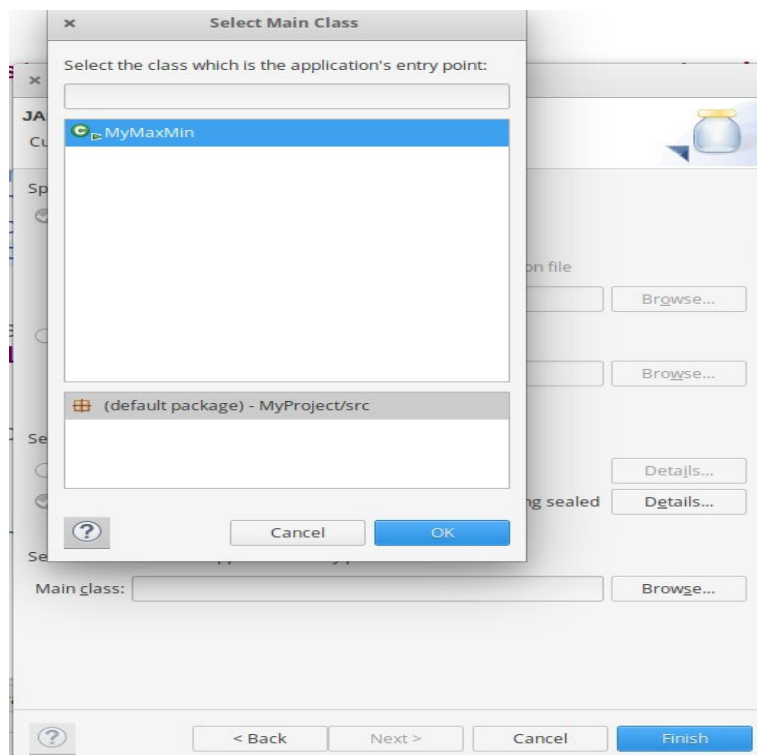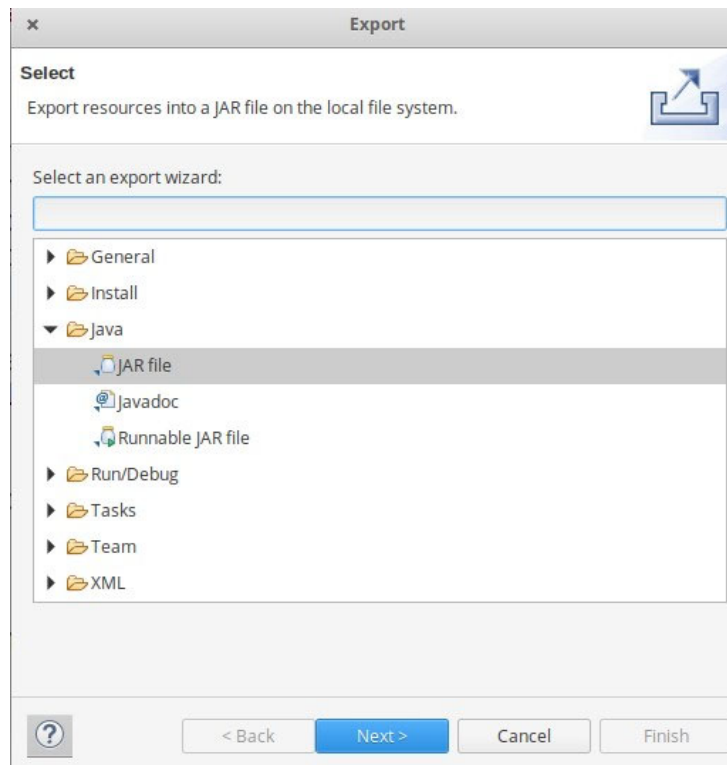
Step 4: Start our Hadoop Daemons
Command

start-dfs.sh
start-yarn.sh

Step 5: Move your dataset to the Hadoop HDFS.
Syntax:
hdfs dfs -put /file_path /destination

In below command / shows the root directory of our HDFS.

hdfs dfs -put /home/dikshant/Downloads/CRND0103-2020-AK_Fairbanks_11_NE.txt /

Check the file sent to our HDFS.

hdfs dfs -ls /

```
dikshant@dikshant-Inspiron-5567:~$ hdfs dfs -put /home/dikshant/Downloads/CRND0103-2020-AK_Fairbanks_11_NE
.txt /
dikshant@dikshant-Inspiron-5567:~$ hdfs dfs -ls /
Found 4 items
-rw-r--r--   1 dikshant supergroup     39711 2020-07-04 09:39 /CRND0103-2020-AK_Fairbanks_11_NE.txt
drwxrwxr-x+  - dikshant supergroup         0 2020-06-23 14:23 /Hadoop_File
drwxrwxrwx   - dikshant supergroup         0 2020-06-14 21:43 /tmp
drwxr-xr-x   - dikshant supergroup         0 2020-06-14 21:43 /user
dikshant@dikshant-Inspiron-5567:~$
```


Step 6: Now Run your Jar File with below command and produce the output in **MyOutput**
File.
Syntax: **hadoop jar /jar_file_location /dataset_location_in_HDFS /output-file_name**
Command:
hadoop jar /home/dikshant/Documents/Project.jar /CRND0103-2020-AK_Fairbanks_11_NE.txt
/MyOutput

```
dikshant@dikshant-Inspiron-5567:~$ hadoop jar /home/dikshant/Documents/Project.jar /CRND0103-2020-AK_Fairb
anks_11_NE.txt /MyOutput
20/07/04 09:44:40 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.sessi
on-id
20/07/04 09:44:40 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
20/07/04 09:44:41 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Im
```


Step 7: Now Move to localhost:50070/, under utilities select Browse the file system and
download **part-r- 00000 in /MyOutput** directory to see result.

		Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
☐		-rw-r--r--	dikshant	supergroup	38.78 KB	Jul 04 09:39	1	122.07 MB	CRND0103-2020-AK_Fairbanks_11_NE.txt	🗑
☐		drwxrwxr-x+	dikshant	supergroup	0 B	Jun 23 14:23	0	0 B	Hadoop_File	🗑
☐		drwxr-xr-x	dikshant	supergroup	0 B	Jul 04 09:44	0	0 B	MyOutput	🗑
☐		drwxrwxrwx	dikshant	supergroup	0 B	Jun 14 21:43	0	0 B	tmp	🗑
☐		drwxr-xr-x	dikshant	supergroup	0 B	Jun 14 21:43	0	0 B	user	🗑

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
☐	-rw-r--r--	dikshant	supergroup	0 B	Jul 04 09:44	1	122.07 MB	_SUCCESS	🗑
☐	-rw-r--r--	dikshant	supergroup	3.85 KB	Jul 04 09:44	1	122.07 MB	part-r-00000	🗑

Step 8: See the result in the Downloaded File.

```
1 The Day is Cold Day :20200101    -21.8
2 The Day is Cold Day :20200102    -23.4
3 The Day is Cold Day :20200103    -25.4
4 The Day is Cold Day :20200104    -26.8
5 The Day is Cold Day :20200105    -28.8
6 The Day is Cold Day :20200106    -30.0
7 The Day is Cold Day :20200107    -31.4
8 The Day is Cold Day :20200108    -33.6
9 The Day is Cold Day :20200109    -26.6
```

In the above image, you can see the top 10 results showing the cold days. The second column is a day in yyyy/mm/dd format.

For Example, **20200101** means , year = 2020 month = 01

Date = 01

c. Patent data files consist of patent id and sub patent id. One patent is associated with multiple sub patents. Write a map reduce code to find out the total sub patent associated with the patent.

package in.project.mapreduce;
import java.io.IOException;
import java.util.StringTokenizer;
/* All org.apache.hadoop packages can be imported using the jar present in lib directory of this java project. */
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Mapper.Context;
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;


/* The Patent program finds the number of sub-patents associated with each id in the provided
input file. We write a map reduce code to achieve this, where mapper makes key value pair
from the input file and reducer does
aggregation on this key value pair. */
public class Patent {
/*Map class is static and extends MapReduceBase and implements Mapper interface having
four hadoop generics type LongWritable, Text, Text, Text. */

public static class Map extends
Mapper<LongWritable, Text, Text, Text> {

//Mapper

/*This method takes the input as text data type and and tokenizes input by taking whitespace as
delimiter. Now key value pair is made and this key value pair is passed to reducer.
@method_arguments key, value, output, reporter @return void */

//Defining a local variable K of type Text
Text k= new Text();

//Defining a local variable v of type Text
Text v= new Text();

@Override
public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

//Converting the record (single line) to String and storing it in a String variable line String line =
value.toString();

//StringTokenizer is breaking the record (line) according to the delimiter whitespace
StringTokenizer tokenizer = new StringTokenizer(line," ");

//Iterating through all the tokens and forming the key value pair

while (tokenizer.hasMoreTokens()) {
```

```
/* The first token is going in jiten, second token in jiten1, third token in jiten,fourth token in
jiten1 and so on. */
String jiten= tokenizer.nextToken();
k.set(jiten);
String jiten1= tokenizer.nextToken();
v.set(jiten1);

//Sending to output collector which inturn passes the same to reducer context.write(k,v);
}
}
}




//Reducer
/* Reduce class is static and extends MapReduceBase and implements Reducer interface having
four hadoop generics type Text, Text, Text, IntWritable. */
public static class Reduce extends Reducer<Text, Text, Text, IntWritable> {
@Override public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {

//Defining a local variable sum of type int int sum = 0;

/* Iterates through all the values available with a key and add them together and give the final
result as the key and sum of its values */


for(Text x : values)
{
sum++;
}

//Dumping the output in context object context.write(key, new IntWritable(sum));
}
}

/*Driver
\* This method is used for setting all the configuration properties. It acts as a driver for map
reduce code.
@return void, @method_arguments args, @throws Exception */

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

//reads the default configuration of cluster from the configuration xml files
Configuration conf = new Configuration();

//Initializing the job with the default configuration of the cluster
Job job = new Job(conf, "patent"); //Assigning the driver class name
```

```
job.setJarByClass(Patent.class);

//Defining the mapper class name
job.setMapperClass(Map.class);

//Defining the reducer class name
job.setReducerClass(Reduce.class);

//Explicitly setting the out key/value type from the mapper if it is not same as that of reducer
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);

//Defining the output key class for the final output i.e. from reducer
job.setOutputKeyClass(Text.class);

//Defining the output value class for the final output i.e. from reducer
job.setOutputValueClass(IntWritable.class);

//Defining the output key class for the final output i.e. from reducer
job.setOutputKeyClass(Text.class);

//Defining the output value class for the final output i.e. from reducer
job.setOutputValueClass(Text.class);

//Defining input Format class which is responsible to parse the dataset into a key value pair
job.setInputFormatClass(TextInputFormat.class);

//Defining output Format class which is responsible to parse the final key-value output from
MR framework to a text file into the hard disk
job.setOutputFormatClass(TextOutputFormat.class);
//setting the second argument as a path in a path variable
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 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);
}
}
```

Output:

Patent	Number of Associated Sub-patents
1	13
2	10
3	4

Practical 3

Aim:- Write a word count program using partitioner and combiner

Using Practitioner

Input: aa bb cc dd ee aa ff bb cc dd ee ff

Save the input as input.txt and place it in Hadoop library.
$vim input.txt
aa bb cc dd ee aa ff bb cc dd ee ff $hadoop fs –mkdir –p /user/rahul/wc/input
$hadoop fs –put input.txt /user/rahul/wc/input

Program:
```
package org.example.wordcount;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Intwritable;
import org.apache.hadoop.io.Text;
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.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;
import org.apache.hadoop.util.GenericOptionParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordcountExample { public int run(String[ ] args) throws Exception {
if(args.length != 2) {
System.out.println("Usage : [input] [output]");
System.exit(-1);
}

Job jobex = jobex.getInstance(getConf());
Jobex.setJobName("wordcount");
Jobex.setJarByClass(WordcountDriver.class);
Jobex.setOutputKeyClass(Text.class);
Jobex.setOutputValueClass(InWritable.class);
Jobex.setMapperClass(WordcountExampleMapper.class);
Jobex.setCombinerClass(WordcountExampleReducer.class);
Jobex.setPartitionerClass(WordcountExamplepartitioner.class);
Jobex.setNumReduceTasks(2);
```

```
Jobex.setReducerClass(WordcountExampleReducer.class);
Jobex.setInputFormatClass(TextInputFormat.class);
Jobex.setOutputFormatClass(TextOutputFormat.class);

Path inputFilePath=new Path(args[0]);
Path outputFilePath=new Path(args[1]); /* This line is to accept the input recursively */
FileInputFormat.setInputDirRecursive(j ob, true);
FileInputFormat.addInputPath(job, inputFilePath);
FileOutputFormat.setOutputPath(job, outputFilePath);
```

/* Delete output file path if already exists */

```
FileSystem fs = FileSystem.newInstance(getConf());
if(fs.exists(outputFilePath)) {
fs.delete(outputFilePath, true);
}
 return jobex.waitForCompletion(true) ? 0: 1; }
}


public class WordcountExampleMapper extends 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, Context context) throws IOException,
InterruptedException {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
word.set(tokenizer.nextToken());
context.write(word, one);
}
}
}


public class WordcountExamplepartitioner extends Partitioner<Text, IntWritable> {
String partitionkey;
public int getPartition(Text key, IntWritable value, int numPartitions) {
// TO DO Auto generated method stub
if(numPartitions = = 2) {
String partitionKey = key.toString();
if(partitoinKey.charAt(0) > „b‟ )
return 0;
else
return 1;
```

```
            }else if(numPartitions = =1)
                    return 0;
        else {
        System.err.println("WordCOuntExamplePartitioner can only handle either 1 or 2
partitions");
        return 0;
}
    } }
public class WordcountExampleReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
private IntWritable totalWordCount = new IntWritable();
public void reduce(final Text key, final Iterable<IntWritable> values, final Context context)
throws
IOException, InterruptedExpection {
int totalcount = 0;
Iterator<IntWritable> iterator = values.iterator();
while (iterator.hasNext()) {
totalcount +=iterator.next().get();
    } totalWordCount.set(totalcount);
    context.write(key, totalWordCount); }
}
```

Output:

```
aa    2

bb    2

cc    2

dd    2

ee    2

ff    2
```

Using Combiner

Input: aa bb cc dd ee aa ff bb cc dd ee ff

Save the input as input.txt and place it in Hadoop library.
$vim input.txt
aa bb cc dd ee aa ff bb cc dd ee ff
$hadoop fs –mkdir –p /user/rahul/wc/input
$hadoop fs –put input.txt /user/rahul/wc/input

Program:
```
package org.example.wordcount;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
```

```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Intwritable;
import org.apache.hadoop.io.Text;
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.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;
import org.apache.hadoop.util.GenericOptionParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordcountExample { public int run(String[ ] args) throws Exception {
if(args.length != 2) {
System.out.println("Usage : [input] [output]");
System.exit(-1);
}
Job jobex = jobex.getInstance(getConf());
Jobex.setJobName("wordcount");
Jobex.setJarByClass(WordcountDriver.class);
Jobex.setOutputKeyClass(Text.class);
Jobex.setOutputValueClass(InWritable.class);
Jobex.setMapperClass(WordcountExampleMapper.class);
Jobex.setCombinerClass(WordcountExampleReducer.class);
Jobex.setReducerClass(WordcountExampleReducer.class);
Jobex.setInputFormatClass(TextInputFormat.class);
Jobex.setOutputFormatClass(TextOutputFormat.class);

Path inputFilePath=new Path(args[0]);
Path outputFilePath=new Path(args[1]);

/* This line is to accept the input recursively */
FileInputFormat.setInputDirRecursive(job, true);
FileInputFormat.addInputPath(job, inputFilePath);
FileOutputFormat.setOutputPath(job, outputFilePath);
/* Delete output file path if already exists */

FileSystem fs = FileSystem.newInstance(getConf());
if(fs.exists(outputFilePath)) { fs.delete(outputFilePath, true); }

return jobex.waitForCompletion(true) ? 0: 1; }
}

public class WordcountExampleMapper extends 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, Context context) throws IOException,
InterruptedException {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
word.set(tokenizer.nextToken());
context.write(word, one);
}
}
}


public class WordcountExampleReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{ private IntWritable totalWordCount = new IntWritable();
public void reduce(final Text key, final Iterable<IntWritable> values, final Context context)
throws IOException, InterruptedExpection { int totalcount = 0;
Iterator<IntWritable> iterator = values.iterator(); while (iterator.hasNext()) {
totalcount +=iterator.next().get();
    }
     totalWordCount.set(totalcount);
    context.write(key, totalWordCount);
    }
}
```

Output:

```
aa   2
bb   2
cc   2
dd   2
ee   2
ff   2
```

Practical 4

Aim:- Configure multimode Hadoop Cluster.

Pre-requisite:
OS: UBUNTU 14.04 LTS
FRAMEWORK: Hadoop 2.7.3
JAVA VERSION: 1.7.0_131

Single node Cluster:

Steps:

1.   check if linux repository service is working or not:
Gcet@gfl1-5:~$ sudo apt-get update

```
Ign http://extras.ubuntu.com trusty InRelease
Ign http://in.archive.ubuntu.com trusty InRelease
Get:1 http://extras.ubuntu.com trusty Release.gpg [72 B]
```

```
Hit http://in.archive.ubuntu.com trusty/universe Translation-en
Ign http://in.archive.ubuntu.com trusty/main Translation-en_IN
Ign http://in.archive.ubuntu.com trusty/multiverse Translation-en_IN
Ign http://in.archive.ubuntu.com trusty/restricted Translation-en_IN
Ign http://in.archive.ubuntu.com trusty/universe Translation-en_IN
Fetched 4,302 kB in 40s (107 kB/s)
Reading package lists... Done
```

2.   Check java version:
Gcet@gfl1-5:~$ java -version
```
java version "1.7.0_131"
OpenJDK Runtime Environment (IcedTea 2.6.9) (7u131-2.6.9-0ubuntu0.14.04.2) OpenJDK
Server VM (build 24.131-b00, mixed mode)
```
3.   Download Hadoop from apache.hadoop.org site and to install hadoop perform the step as under:
Gcet@gfl1-5:~$ tar -xvf Hadoop-2.7.3.tar.gz hadoop-2.7.3/share/hadoop/tools/lib/hadoop-extras-2.7.3.jar hadoop-2.7.3/share/hadoop/tools/lib/asm-3.2.jar hadoop-2.7.3/include/ hadoop-2.7.3/include/hdfs.h hadoop-2.7.3/include/Pipes.hh hadoop-2.7.3/include/TemplateFactory.hh hadoop-2.7.3/include/StringUtils.hh hadoop-2.7.3/include/SerialUtils.hh hadoop-2.7.3/LICENSE.txt hadoop-2.7.3/NOTICE.txt hadoop-2.7.3/README.txt
Gcet@gfl1-5:~$ sudo mv/home/Gcet/Downloads/Hadoop-2.7.3 /usr/local/Hadoop

4.   check if hadoop is working properly or not using the command under:
Gcet@gfl1-$ /usr/local/hadoop/hadoop-2.7.3/bin/hadoop

classpath   prints the class path needed to get the credential   interact with credential providers
Hadoop jar and the required libraries
daemonlog   get/set the log level for each daemon trace   view and modify Hadoop tracing settings
Most commands print help when invoked w/o parameters.

5.   install openssl, ssh and rsync:

Gcet@gfl1-5:~$ sudo apt-get install openssl

[sudo] password for Gcet:
Reading package lists... Done
Building dependency tree
Reading state information... Done openssl is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and
460 not upgraded.

Gcet@gfl1-5:~$ sudo apt-get install ssh

Reading package lists... Done
Building dependency tree
Reading state information... Done ssh is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 460 not upgraded.

Gcet@gfl1-5:~$ sudo apt-get install ssl

Reading package lists... Done
Building dependency tree
Reading state information... Done rsync is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and
460 not upgraded.

6.   set environment variable for java:
Gcet@gfl1-5:~$ export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-i386

7.   to run examples on single system requires to create input file

Gcet@gfl1-5:~$ mkdir input1
Gcet@gfl1-5:~$ mkdir output1

8.   Now copy xml file from Hadoop folder to input folder

Gcet@gfl1-5:~$ cp /usr/local/hadoop/hadoop-2.7.3/etc/hadoop/capacity-scheduler.xml /home/Gcet/Desktop/input1

9.   Now run Hadoop examples:
Gcet@gfl1-5:~$ /usr/local/hadoop/hadoop-2.7.3/bin/hadoop jar /usr/local/hadoop/hadoop-2.7.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar grep /home/Gcet/Desktop/input1/ /home/Gcet/output1/output1 'principal[.]*'

17/07/26 15:15:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/07/26 15:15:52 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
17/07/26 15:15:52 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
17/07/26 15:15:52 INFO input.FileInputFormat: Total input paths to process : 2
17/07/26 15:15:52 INFO mapreduce.JobSubmitter: number of splits:2
17/07/26 15:15:53 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1790612813_0001
17/07/26 15:15:53 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
17/07/26 15:15:53 INFO mapreduce.Job: Running job: job_local1790612813_0001
17/07/26 15:15:53 INFO mapred.LocalJobRunner: OutputCommitter set in config null
17/07/26 15:15:53 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/07/26 15:15:53 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
17/07/26 15:15:53 INFO mapred.LocalJobRunner: Waiting for map tasks 17/07/26 15:15:53 INFO mapred.LocalJobRunner: Starting task: attempt_local1790612813_0001_m_000000_0
17/07/26 15:15:53 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/07/26 15:15:53 INFO mapred.Task: Using ResourceCalculatorProcessTree : [ ]
.
.
. .
17/07/26 15:15:55 INFO mapreduce.Job: Job job_local192240145_0002 running in uber mode : false
17/07/26 15:15:55 INFO mapreduce.Job: map 100% reduce 100%
17/07/26 15:15:55 INFO mapreduce.Job: Job job_local192240145_0002 completed successfully 17/07/26 15:15:55 INFO mapreduce.Job: Counters: 30 File System Counters
FILE: Number of bytes read=1195494
FILE: Number of bytes written=2315812
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
Map-Reduce Framework
Map input records=0
Spilled Records=0
Shuffled Maps =1
GC time elapsed (ms)=10
Total committed heap usage (bytes)=854065152
ShuffleBAD_ID=0 CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0 WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
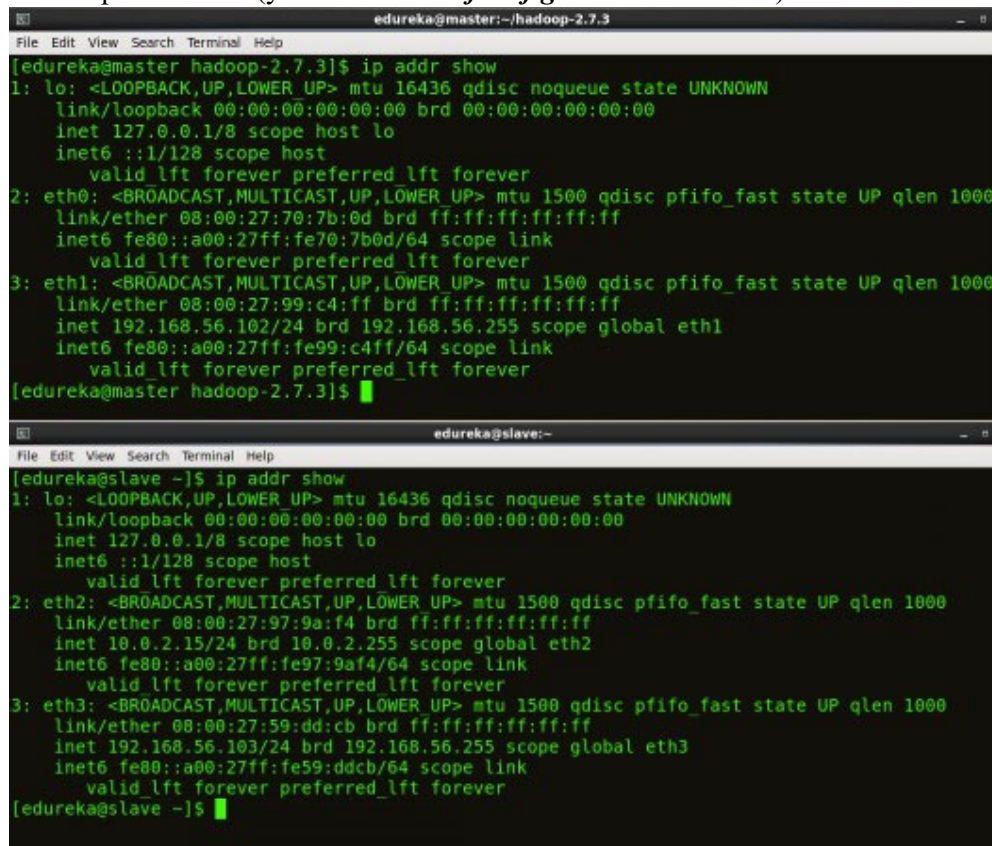
Multi node cluster:

Steps:
Bytes Read=98
File Output Format Counters Bytes
Written=8

We have two machines (master and slave) with IP:
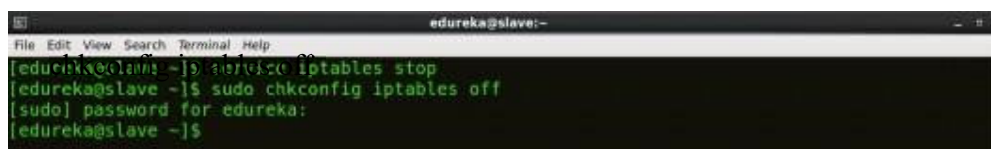
Master IP: **192.168.56.102**

Slave IP: **192.168.56.103**

STEP 1: Check the IP address of all machines.

Command: ip addr show (you can use the *ifconfig* command as well)

STEP 2: Disable the firewall restrictions. *Command:* service iptables stop

Command: sudo
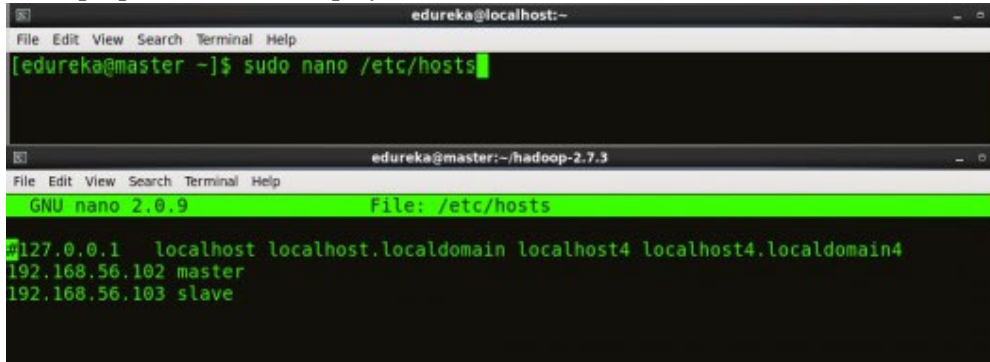
STEP 3: Open hosts file to add master and data node with their respective IP addresses.

Command: sudo nano /etc/hosts
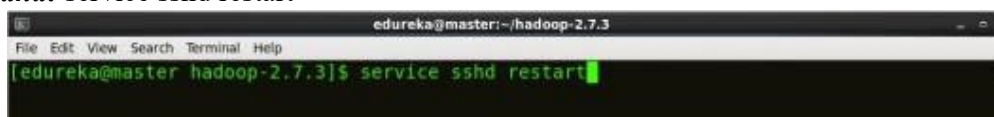
Same properties will be displayed in the master and slave hosts files.

STEP 4: Restart the sshd service.
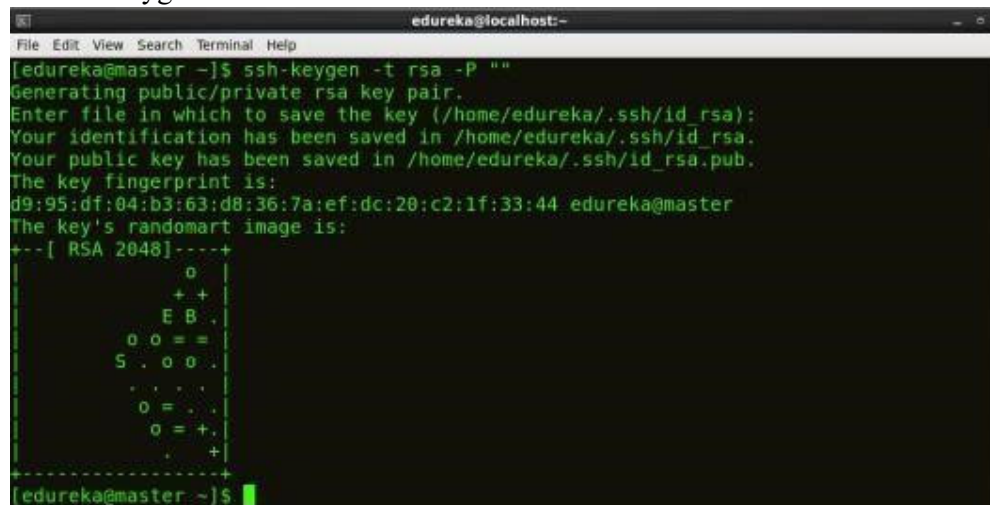
Command: service sshd restart

STEP 5: Create the SSH Key in the master node. (Press enter button when it asks you to enter a filename to save the key).
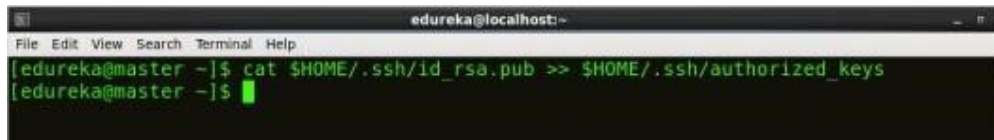
Command: ssh-keygen -t rsa -P ""

STEP 6: Copy the generated ssh key to master node's authorized keys.
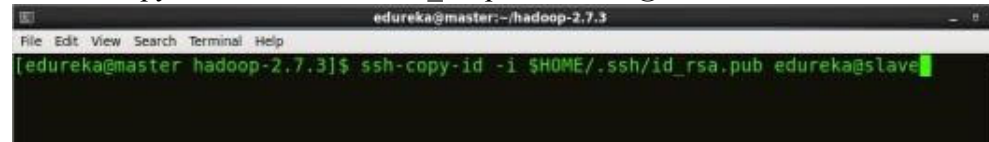
Command: cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

STEP 7: Copy the master node's ssh key to slave's authorized keys.

Command: ssh-copy-id -i $HOME/.ssh/id_rsa.pub edureka@slave

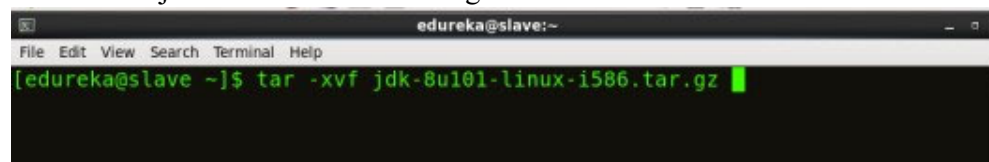**STEP 8:** Click here to download the Java 8 Package. Save this file in your home directory.

STEP 9: Extract the Java Tar File on all nodes.
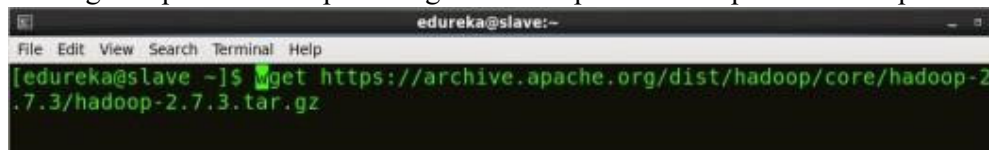
Command: tar -xvf jdk-8u101-linux-i586.tar.gz

STEP 10: Download the Hadoop 2.7.3 Package on all nodes.
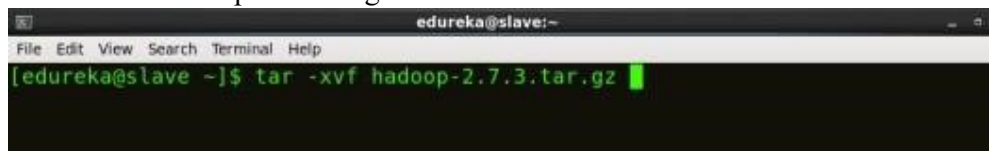
Command: wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz
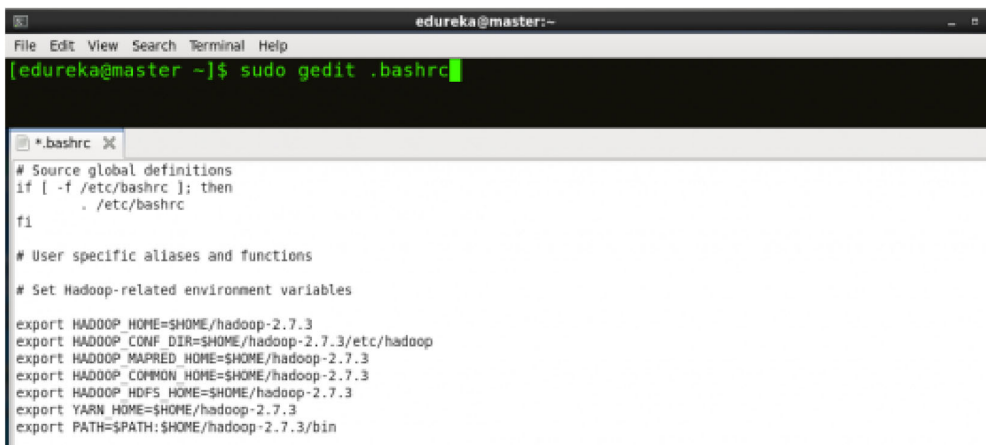
STEP 11: Extract the Hadoop tar File on all nodes.

Command: tar -xvf hadoop-2.7.3.tar.gz

STEP 12: Add the Hadoop and Java paths in the bash file (.bashrc) on all nodes. Open**. bashrc**

file. Now, add Hadoop and Java Path as shown below: *Command*: sudo gedit .bashrc
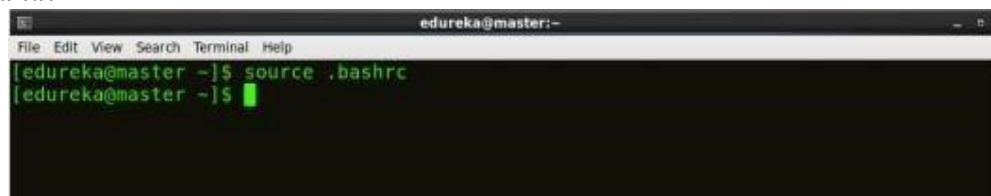
Then, save the bash file and close it.

For applying all these changes to the current Terminal, execute the source command.
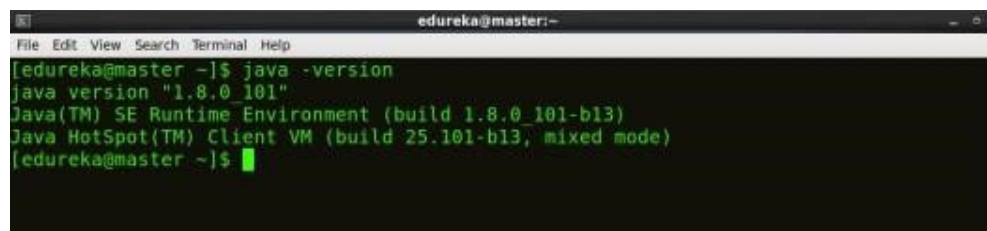
Command: source .bashrc
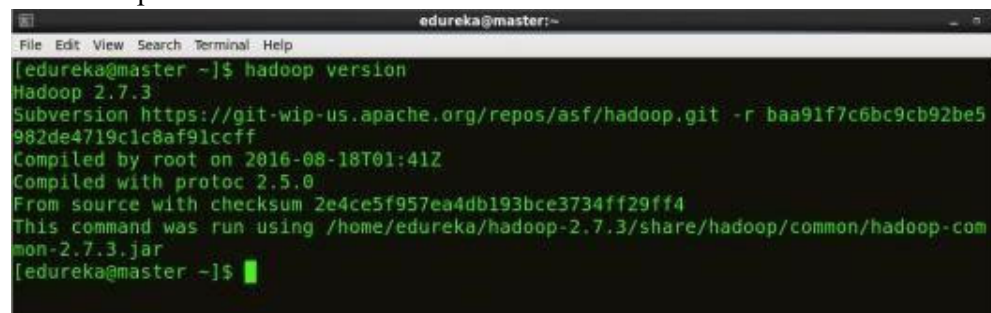
To make sure that Java and Hadoop have been properly installed on your system and can be accessed through the Terminal, execute the java -version and hadoop version commands.

Command: java -version

Command: hadoop version

Now edit the configuration files in **hadoop-2.7.3/etc/hadoop** directory.

STEP 13: Create masters file and edit as follows in both master and slave machines as below:

Command: sudo gedit masters
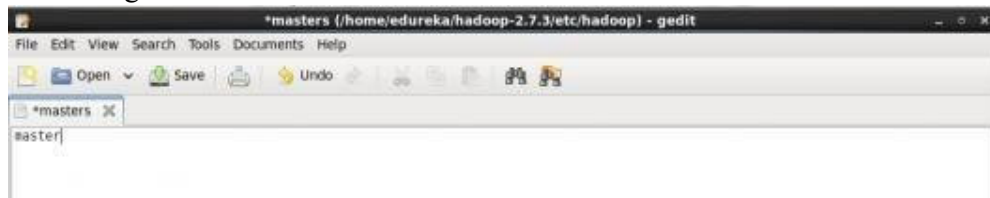
STEP 14: Edit slaves file in master machine as follows:

Command: sudo gedit /home/edureka/hadoop-2.7.3/etc/hadoop/slaves
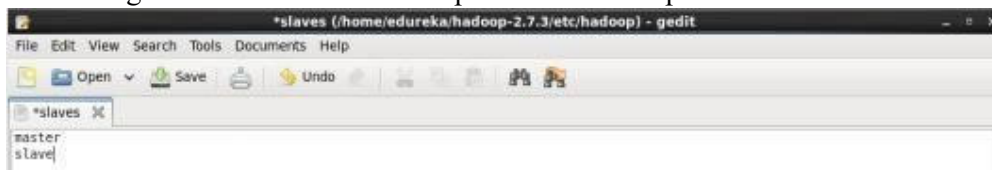
STEP 15: Edit slaves file in slave machine as follows:

Command: sudo gedit /home/edureka/hadoop-2.7.3/etc/hadoop/slaves
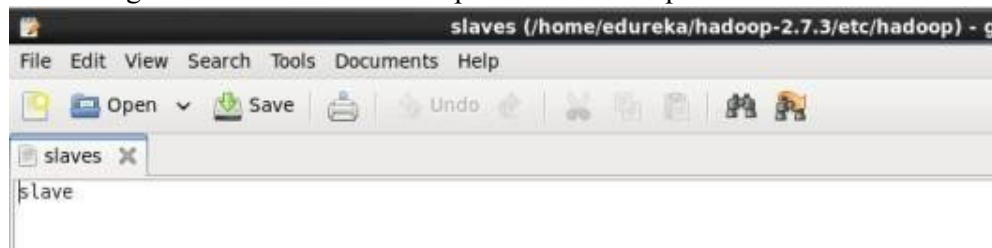
STEP 16: Edit core-site.xml on both master and slave machines as follows:

Command: sudo gedit /home/edureka/hadoop-2.7.3/etc/hadoop/core-site.xml
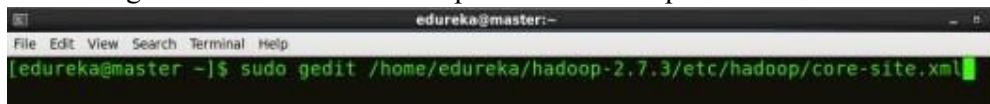

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3 <configuration>
4 <property>
5 <name>fs.default.name</name>
6 <value>hdfs://master:9000</value>
7 </property>
8 </configuration>
```

STEP 7: Edit hdfs-site.xml on master as follows:

Command: sudo gedit /home/edureka/hadoop-2.7.3/etc/hadoop/hdfs-site.xml
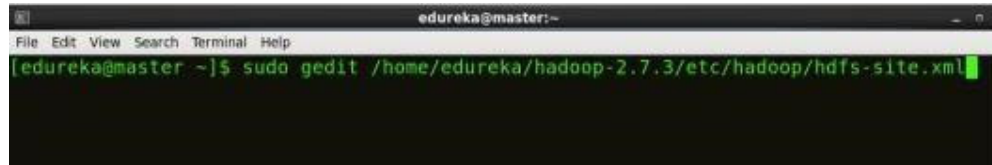

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3  <configuration>
4  <property>
5  <name>dfs.replication</name>
6  <value>2</value>
7  </property>
8  <property>
9  <name>dfs.permissions</name>
10<value>false</value>
11</property>
12<property>
13<name>dfs.namenode.name.dir</name>
14<value>/home/edureka/hadoop-2.7.3/namenode</value>
15</property>
16<property>
17<name>dfs.datanode.data.dir</name>

18<value>/home/edureka/hadoop-2.7.3/datanode</value>
19</property>
20</configuration>
```

STEP 18: Edit hdfs-site.xml on slave machine as follows:

Command: sudo gedit /home/edureka/hadoop-2.7.3/etc/hadoop/hdfs-site.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3  <configuration>
4  <property>
5  <name>dfs.replication</name>
6  <value>2</value>
7  </property>
8  <property>
9  <name>dfs.permissions</name>
10<value>false</value>
11</property>
12<property>
```
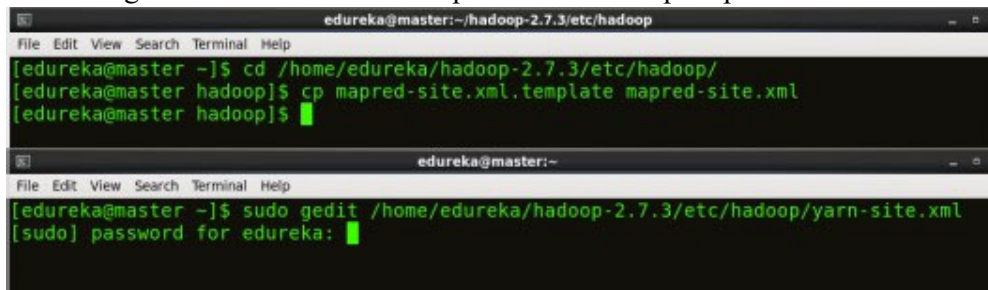
13 <name>dfs.datanode.data.dir</name>
14 <value>/home/edureka/hadoop-2.7.3/datanode</value>
15 </property>
16 </configuration>

STEP 19: Copy mapred-site from the template in configuration folder and the edit mapred-

site.xml on both master and slave machines as follows:

Command: cp mapred-site.xml.template mapred-site.xml

Command: sudo gedit /home/edureka/hadoop-2.7.3/etc/hadoop/mapred-site.xml

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3 <configuration>
4 <property>
5 <name>mapreduce.framework.name</name>
6 <value>yarn</value>
7 </property>
8 </configuration>

STEP 20: Edit yarn-site.xml on both master and slave machines as follows:
Command: sudo gedit /home/edureka/hadoop-2.7.3/etc/hadoop/yarn-site.xml
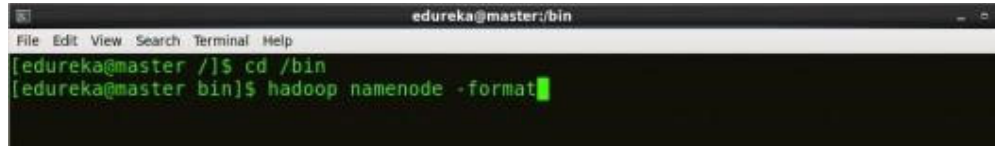
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3 <configuration>
4 <property>
5 <name>yarn.nodemanager.aux-services</name>
6 <value>mapreduce_shuffle</value>
7 </property>
8 <property>
9 <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
10 <value>org.apache.hadoop.mapred.ShuffleHandler</value>
11 </property>

12 </configuration>

STEP 21: Format the namenode (Only on master machine).
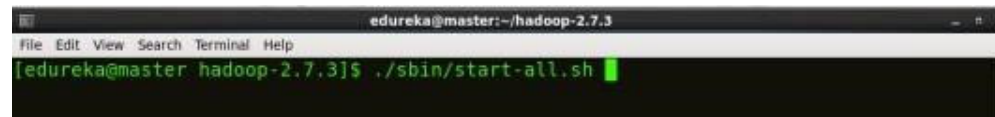
Command: hadoop namenode -format

STEP 22: Start all daemons (Only on master machine).
Command: ./sbin/start-all.sh
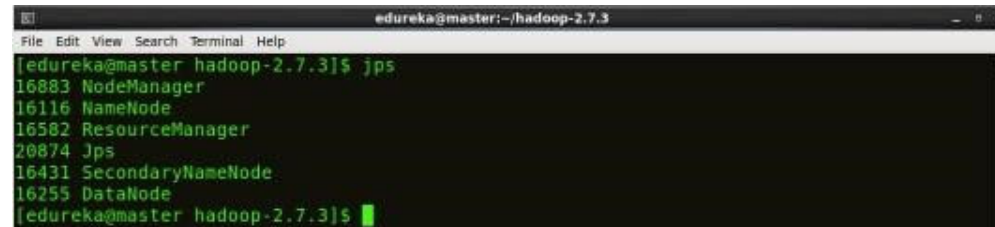
STEP 23: Check all the daemons running on both master and slave machines.
Command: jps
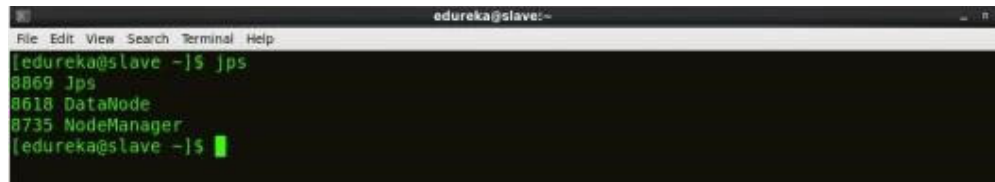
On master

On slave

At last, open the browser and go to **master**:**50070/dfshealth.html** on your master machine, this will give you the NameNode interface. Scroll down and see for the number of **live nodes**, if its **2**, you have successfully setup a multi node Hadoop cluster. In case, it's not 2, you might have missed out any of the steps which I have mentioned above. But no need to worry, you can go back and verify all the configurations again to find the issues and then correct them.

Configured Capacity:	34.47 GB
DFS Used:	48 KB (0%)
Non DFS Used:	17.15 GB
DFS Remaining:	17.32 GB (50.25%)
Block Pool Used:	48 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	2 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0

Practical 5

Aim:- Configure Sqoop. Try sqoop import and export command.

Step 1: Verifying JAVA Installation

You need to have Java installed on your system before installing Sqoop.

$ 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 on your system, then follow the steps given below.

Download Java (JDK <latest version> - X64.tar.gz) by visiting the following link.

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

Step 2: Verifying Hadoop Installation

Hadoop must be installed on your system before installing Sqoop.

$ 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.

Step 3: Downloading Sqoop

We can download the latest version of Sqoop from the following link For this tutorial, we are using version 1.4.5, that is, **sqoop-1.4.5.bin_hadoop-2.0.4-alpha.tar.gz.**

Step 4: Installing Sqoop

The following commands are used to extract the Sqoop tar ball and move it to "/usr/lib/sqoop" directory.
$tar -xvf sqoop-1.4.4.bin_hadoop-2.0.4-alpha.tar.gz
$ su password:

mv sqoop-1.4.4.bin_hadoop-2.0.4-alpha /usr/lib/sqoop #exit

Step 5: Configuring bashrc

You have to set up the Sqoop environment by appending the following lines to ~/.bashrc file −

#Sqoop
export SQOOP_HOME=/usr/lib/sqoop export PATH=$PATH:$SQOOP_HOME/bin

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

$ source ~/.bashrc

Step 6: Configuring Sqoop

To configure Sqoop with Hadoop, you need to edit the **sqoop-env.sh** file, which is placed in the **$SQOOP_HOME/conf** directory.

First of all, Redirect to Sqoop config directory and copy the template file using the following command:-

$ cd $SQOOP_HOME/conf
$ mv sqoop-env-template.sh sqoop-env.sh

Open **sqoop-env.sh** and edit the following lines:-

export HADOOP_COMMON_HOME=/usr/local/hadoop export
HADOOP_MAPRED_HOME=/usr/local/hadoop

Step 7: Download and Configure mysql-connector-java

We can download **mysql-connector-java-5.1.30.tar.gz** file from the following link.
http://ftp.ntu.edu.tw/MySQL/Downloads/Connector-J/

The following commands are used to extract mysql-connector-java tarball and move **mysql-connector- java-5.1.30-bin.jar** to /usr/lib/sqoop/lib directory.

$ tar -zxf mysql-connector-java-5.1.30.tar.gz
$ su password:

cd mysql-connector-java-5.1.30

mv mysql-connector-java-5.1.30-bin.jar /usr/lib/sqoop/lib

Step 8: Verifying Sqoop

The following command is used to verify the Sqoop version.

$ cd $SQOOP_HOME/bin
$ sqoop-version
Expected output −

30/6/23 10:25:13 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
Sqoop 1.4.5 git commit id 5b34accaca7de251fc91161733f906af2eddbe83
Sqoop Import & Export Command

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**.

Sqoop Import

The following syntax is used to import data into HDFS.

$ sqoop import (generic-args) (import-args)
$ sqoop-import (generic-args) (import-args)

Importing a Table

Sqoop tool „import" is used **to import table data from the table to the Hadoop file system** as **a text file or a binary file.**

The following command is used to import the emp table from MySQL database server to HDFS.

**$ sqoop import **
**--connect jdbc:mysql://localhost/userdb **
**--username root **
--table emp --m 1

If it is executed successfully, then you get the following output.

30/06/23 10:25:34 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
30/06/23 10:25:34 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset. 30/06/23 10:25:34 INFO tool.CodeGenTool: Beginning code generation 30/06/23 10:25:39 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `emp` AS t LIMIT 1
30/06/23 10:25:39 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `emp` AS t LIMIT 1
30/06/23 10:25:54 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/local/hadoop 30/06/23 10:26:34 INFO orm.CompilationManager: Writing jar file:

/tmp/sqoop-hadoop/compile/cebe706d23ebb1fd99c1f063ad51ebd7/emp.jar

    ----

    30/06/23 10:26:42 INFO mapreduce.Job: The url to track the job:
    http://localhost:8088/proxy/application_1419242001831_0001/
  30/06/23 10:27:14 INFO mapreduce.Job: Job job_1419242001831_0001 running in uber mode :
    false
    30/06/23 10:27:44 INFO mapreduce.Job: map 0% reduce 0%
    30/06/23 10:28:10 INFO mapreduce.Job: map 100% reduce 0%
    30/06/23 10:28:18 INFO mapreduce.Job: Job job_1419242001831_0001 completed successfully

    ----

  30/06/23 10:28:22 INFO mapreduce.ImportJobBase: Transferred 145 bytes in 177.5849 seconds
    (0.8165 bytes/sec)
    30/06/23 10:25:23 INFO mapreduce.ImportJobBase: Retrieved 5 records.

To verify the imported data in HDFS, use the following command.

$ $HADOOP_HOME/bin/hadoop fs -cat /emp/part-m-*

It shows you the emp table data and fields are separated with comma (,).

1201, gopal, manager, 50000, TP
1202, manisha, preader, 50000, TP
1203, kalil, php dev, 30000, AC
1204, prasanth, php dev, 30000, AC
1205, kranthi, admin, 20000, TP

Importing into Target Directory

We can specify the target directory while importing table data into HDFS using the Sqoop import tool.

Following is the syntax to specify the target directory as option to the Sqoop import command.

--target-dir <new or exist directory in HDFS>

The following command is used to import **emp_add** table data into „/queryresult‟ directory.

**$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp_add \
--m 1 \
--target-dir /queryresult**

The following command is used to verify the imported data in /queryresult directory form emp_add table.

$ $HADOOP_HOME/bin/hadoop fs -cat /queryresult/part-m-*

It will show you the emp_add table data with comma (,) separated fields.

1201, 288A, vgiri, jublee
1202, 108I, aoc, sec-bad
1203, 144Z, pgutta, hyd
1204, 78B, oldcity, sec-bad
1205, 720C, hitech, sec-bad

Import Subset of Table Data

We can import a subset of a table using the „where" clause in Sqoop import tool. It executes the corresponding SQL query in the respective database server and stores the result in a target directory in HDFS.

The syntax for where clause is as follows.

--where <condition>

The following command is used to import a subset of emp_add table data. The subset query is to retrieve the employee id and address, who lives in Secunderabad city.

**$ sqoop import **
**--connect jdbc:mysql://localhost/userdb **
**--username root **
**--table emp_add **
**--m 1 **
**--where "city ="sec-bad"" **
--target-dir /wherequery

The following command is used to verify the imported data in /wherequery directory from the **emp_add** table.

$ $HADOOP_HOME/bin/hadoop fs -cat /wherequery/part-m-*

It will show you the emp_add table data with comma (,) separated fields.

1202, 108I, aoc, sec-bad
1204, 78B, oldcity, sec-bad
1205, 720C, hitech, sec-bad

Incremental Import

Incremental import is a technique that imports only the newly added rows in a table. It is required to add „incremental", „check-column", and „last-value" options to perform the incremental import.

The following syntax is used for the incremental option in Sqoop import command**.**

--incremental <mode>
--check-column <column name>
--last value <last check column value>

Let us assume the newly added data into **emp** table is as follows:-

1206, satish p, grp des, 20000, GR

The following command is used to perform the incremental import in the **emp** table.

**$ sqoop import **
**--connect jdbc:mysql://localhost/userdb **
**--username root **
**--table emp **
**--m 1 **
**--incremental append **
**--check-column id **
-last value 1205

The following command is used to verify the imported data from **emp** table to HDFS emp/ directory.

$ $HADOOP_HOME/bin/hadoop fs -cat /emp/part-m-*
It shows you the emp table data with comma (,) separated fields.

1201, gopal, manager, 50000, TP
1202, manisha, preader, 50000, TP
1203, kalil, php dev, 30000, AC
1204, prasanth, php dev, 30000, AC
1205, kranthi, admin, 20000, TP
1206, satish p, grp des, 20000, GR

The following command is used to see the **modified or newly added rows** from the **emp** table.

$ $HADOOP_HOME/bin/hadoop fs -cat /emp/part-m-*1

It shows you the **newly added rows** to the **emp** table with comma (,) separated fields.

1206, satish p, grp des, 20000, GR

Sqoop - Import All Tables

Syntax

The following syntax is used to import all tables**.**

$ sqoop import-all-tables (generic-args) (import-args)
$ sqoop-import-all-tables (generic-args) (import-args)

Example: Let us take an example of importing all tables from the userdb database. The list of tables that the database userdb contains is as follows.

The following command is used to import all the tables from the userdb database.

**$ sqoop import-all-tables **
**--connect jdbc:mysql://localhost/userdb **
--username root

Note − If you are using the import-all-tables, it is mandatory that every table in that database must have a primary key field.

The following command is used to verify all the table data to the userdb database in HDFS.

$ $HADOOP_HOME/bin/hadoop fs –ls

It will show you the list of table names in userdb database as directories.

Output

drwxr-xr-x - hadoop supergroup 0 2014-12-22 22:50 _sqoop drwxr-xr-x - hadoop supergroup 0 2014-12-23 01:46 emp drwxr-xr-x - hadoop supergroup 0 2014-12-23 01:50 emp_add drwxr-xr-x - hadoop supergroup 0 2014-12-23 01:52 emp_contact **Sqoop – Export**

Syntax : The following is the syntax for the export command.

$ sqoop export (generic-args) (export-args)
$ sqoop-export (generic-args) (export-args)

Example

Let us take an example of the employee data in file, in HDFS. The employee data is available in **emp_data** file in „emp/‟ directory in HDFS.

The following query is used to create the table „employee‟ in mysql command line.

 $ mysql mysql> USE db;
mysql> CREATE TABLE employee (
id INT NOT NULL PRIMARY KEY, name VARCHAR(20),
deg VARCHAR(20), salary INT, dept VARCHAR(10));

The following command is used to export the table data (which is in **emp_data** file on HDFS) to the employee table in db database of Mysql database server.

**$ sqoop export **
**--connect jdbc:mysql://localhost/db **
**--username root **
**--table employee **
--export-dir /emp/emp_data

The following command is used to verify the table in mysql command line.

mysql>select * from employee;

If the given data is stored successfully, then you can find the following table of given employee data.

Id	Name	Designation	Salary	Dept
1201	gopal	manager	50000	TP
1202	manisha	preader	50000	TP
1203	kalil	php dev	30000	AC
1204	prasanth	php dev	30000	AC
1205	kranthi	admin	20000	TP
1206	satish p	grp des	20000	GR

Sqoop job

Sqoop job creates and saves the import and export commands. It specifies parameters to identify and recall the saved job. This re-calling or re-executing is used in the incremental import, which can import the updated rows from RDBMS table to HDFS.

Syntax : The following is the syntax for creating a Sqoop job.
$ sqoop job (generic-args) (job-args)
[-- [subtool-name] (subtool-args)]

$ sqoop-job (generic-args) (job-args)
[-- [subtool-name] (subtool-args)]

Create Job (--create)

Here we are creating a job with the name myjob, which can import the table data from RDBMS table to HDFS. The following command is used to create a job that is importing data from the employee table in the db database to the HDFS file.

**$ sqoop job --create myjob **
**-- import **
**--connect jdbc:mysql://localhost/db **
**--username root **
--table employee --m 1

Verify Job (--list)
„--list" argument is used to verify the saved jobs. The following command is used to verify the list of saved Sqoop jobs.
$ sqoop job --list

It shows the list of saved jobs.
Available jobs:
Myjob

Inspect Job (--show)
„--show" argument is used to inspect or verify particular jobs and their details. The following command and sample output is used to verify a job called **myjob**.

$ sqoop job --show myjob
It shows the tools and their options, which are used in **myjob.**
Job: myjob
Tool: import Options:

- - - - - - - - - - - - - - - - - - - -·
direct.import = true
codegen.input.delimiters.record = 0 hdfs.append.dir = false db.table = employee
... incremental.last.value = 1206...
Execute Job (--exec)
„--exec" option is used to execute a saved job. The following command is used to execute a saved job called myjob.
$ sqoop job --exec myjob

It shows you the following output.
30/06/23 11:28:50 INFO tool.CodeGenTool: Beginning code generation
...

Practical 6

Aim:- Configure Hive and try basic Hive query.

Step 1: Verifying JAVA Installation

Java must be installed on your system before installing Hive.

$ 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.

Step 2: Verifying Hadoop Installation

Hadoop must be installed on your system before installing Hive.

$ 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 install and configure all steps.

Step 3: Downloading Hive

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 Derby

You 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/derbyt
ools.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/hivecommon-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>

<u>Query in HIVE:</u>

1)   <u>Retrieving information:</u>

SELECT from_columns FROM table WHERE conditions;

2)   All values:
SELECT * FROM tables;

3)   Some Values:
SELECT * FROM table WHERE rec_name = "value";

4)   Multiple Criteria:-
SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2

5)   Selecting specific columns
SELECT column_name FROM table

6)   Retrieving unique output records
SELECT DISTINCT column_name FROM table;

7)   Sorting
SELECT col1, col2 FROM table ORDER BY col2;

8)   Sorting backward
SELECT col1, col2 FROM table ORDER BY col2 DESC;

9)   Counting rows
SELECT COUNT(*) FROM table;

10)   Grouping with counting
SELECT owner, COUNT(*) FROM table GROUP BY owner;

11)   Maximum value
SELECT MAX(col_name) AS label FROM table;

12)   Minimum value
SELECT MIN(col_name) AS label FROM table;

13)   Average Value
SELECT AVG(col_name) AS label FROM table;

14)   Sum Value
SELECT SUM(col_name) AS label FROM table;

15)   Count Value
SELECT COUNT(col_name) AS label FROM table;

16)   Selecting from multiple tables(Join same table using alias w/"AS")
SELECT pet.name, comment FROM pet JOIN event ON(pet.name = event.name)

17)   Selecting a database
USE database;

18)   Listing databases
SHOW DATABASES;

19)   Listing tables in a database
SHOW TABLES;

20)   Describing the format of a table
DESCRIBE (FORMATTED|EXTENDED) table;

21)   Creating a database
CREATE DATABASE db_name;

22)   Dropping a database
DROP DATABASE db_name (CASCADE);

Practical 7

Aim:- Write Hive Query for the following task for movie dataset. Movie dataset consists of movie id, movie name, release year, rating, and runtime in seconds. A sample of the dataset is as follows:
a. The Nightmare Before Christmas,1993,3.9,4568
b. The Mummy,1932,3.5,4388
c. Orphans of the Storm,1921,3.2,9062
d. The Object of Beauty,1991,2.8,6150
e. Night Tide,1963,2.8,5126 Write a hive query for the following a. Load the data
b. List the movies that are having a rating greater than 4
c. Store the result of previous query into file
d. List the movies that were released between 1950 and 1960
e. List the movies that have duration greater than 2 hours
f. List the movies that have rating between 3 and 4
g. List the movie names and its duration in minutes
h. List all the movies in the ascending order of year.
i. List all the movies in the descending order of year.
j. list the distinct records.
k. Use the LIMIT keyword to get only a limited number for results from relation.
l. Use the sample keyword to get a sample set from your data.
m.   View the step-by-step execution of a sequence of statements using ILLUSTRATE command.

Assuming the table name is movies and the columns are movie_name, release_year, rating, and runtime.

   a. Load the data:
 CREATE TABLE movies (movie_name STRING, release_year INT, rating FLOAT, runtime INT);
 LOAD DATA LOCAL INPATH 'path/to/your/movie/data' INTO TABLE movies;

   b.   List movies with rating greater than 4:
 SELECT * FROM movies WHERE rating > 4;

   c.   Store the result into a file:
 INSERT OVERWRITE LOCAL DIRECTORY 'path/to/output/directory' ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
 SELECT * FROM movies WHERE rating > 4;

   d.   List movies released between 1950 and 1960:
 SELECT * FROM movies WHERE release_year BETWEEN 1950 AND 1960;

e.   List movies with duration greater than 2 hours (7200 seconds):
SELECT * FROM movies WHERE runtime > 7200;

f.   List movies with rating between 3 and 4:
SELECT * FROM movies WHERE rating BETWEEN 3 AND 4;

List movie names and duration in minutes:
SELECT movie_name, runtime / 60 AS duration_minutes FROM movies;

g.   List all movies in ascending order of release year:
SELECT * FROM movies ORDER BY release_year ASC;

h.   List all movies in descending order of release year:
SELECT * FROM movies ORDER BY release_year DESC;

i.   List distinct records:
SELECT DISTINCT * FROM movies;

j.   Use the LIMIT keyword to get a limited number of results:
SELECT * FROM movies LIMIT 10;

k.   Use the SAMPLE keyword to get a sample set from the data:
SELECT * FROM movies SAMPLE (10);

l.   View step-by-step execution using ILLUSTRATE command:
SET hive.exec.mode.local.auto=false;
ILLUSTRATE SELECT * FROM movies WHERE rating > 4;

Practical 8

Aim:- Configure Pig and try different Pig commands.

Prerequisites
It is essential that you have Hadoop and Java installed on your system before you go for Apache Pig.

Downloading Pig

First of all, download the latest version of Apache Pig from the following website −
https://pig.apache.org/

Install Apache Pig

Step 1
Create a directory with the name Pig in the same directory where the installation directories of Hadoop, Java, and other software were installed. (We have created the Pig directory in the user named Hadoop).

$ mkdir Pig

Step 2
Extract the downloaded tar files as shown below.

$ cd Downloads/
$ tar zxvf pig-0.15.0-src.tar.gz
$ tar zxvf pig-0.15.0.tar.gz

Step 3
Move the content of pig-0.15.0-src.tar.gz file to the Pig directory created earlier as shown below.

$ mv pig-0.15.0-src.tar.gz/*

/home/Hadoop/Pig/ Configure Apache Pig
After installing Apache Pig, we have to configure it. To configure, we need to edit two files − **bashrc and pig.properties.**

.bashrc file

In the **.bashrc** file, set the following variables −

• **PIG_HOME** folder to the Apache Pig's installation folder,

• **PATH** environment variable to the bin folder, and

• **PIG_CLASSPATH** environment variable to the etc (configuration) folder of your Hadoop

installations (the directory that contains the core-site.xml, hdfs-site.xml and mapred-site.xml files).

export PIG_HOME = /home/Hadoop/Pig
export PATH =
$PATH:/home/Hadoop/pig/bin export PIG_CLASSPATH = $HADOOP_HOME/conf

pig.properties file
In the **conf** folder of Pig, we have a file named **pig.properties.** In the pig.properties file, you can set various parameters as given below.

pig -h properties

The following properties are supported –

Logging: verbose = true|false; default is false. This property is the same as -v switch
brief=true|false; default is false. This property is the same as -b switch
debug=OFF|ERROR|WARN|INFO|DEBUG; default is INFO.
This property is the same as -d switch aggregate.warning = true|false; default is true. If true, prints count of warnings of each type rather than logging each warning.

Performance tuning: pig.cachedbag.memusage=<mem fraction>; default is 0.2 (20% of all memory).
Note that this memory is shared across all large bags used by the application.
pig.skewedjoin.reduce.memusagea=<mem fraction>; default is 0.3 (30% of all memory).
Specifies the fraction of heap available for the reducer to perform the join.
pig.exec.nocombiner = true|false; default is false.
Only disable combiner as a temporary workaround for problems.
opt.multiquery = true|false; multiquery is on by default.
   Only disable multiquery as a temporary workaround for problems. opt.fetch=true|false; fetch is on by default.
   Scripts containing Filter, Foreach, Limit, Stream, and Union can be dumped without MR jobs.
pig.tmpfilecompression = true|false; compression is off by default.
   Determines whether output of intermediate jobs is compressed. pig.tmpfilecompression.codec = lzo|gzip; default is gzip.
Used in conjunction with pig.tmpfilecompression. Defines compression type.
 pig.noSplitCombination = true|false. Split combination is on by default. Determines if multiple small files are combined into a single map.

pig.exec.mapPartAgg = true|false. Default is false.
Determines if partial aggregation is done within map phase, before records are sent to combiner.
pig.exec.mapPartAgg.minReduction=<min aggregation factor>. Default is 10.
       If the in-map partial aggregation does not reduce the output num records by this factor, it gets disabled.

Miscellaneous: exectype = mapreduce|tez|local; default is mapreduce. This property is the same as -x switch pig.additional.jars.uris=<comma seperated list of jars>. Used in place of register

command. udf.import.list=<comma seperated list of imports>. Used to avoid package names in UDF. stop.on.failure = true|false; default is false. Set to true to terminate on the first error.

   pig.datetime.default.tz=<UTC time offset>. e.g. +08:00. Default is the default timezone of the host. Determines the timezone used to handle datetime datatype and UDFs. Additionally, any Hadoop property can be specified.

Verifying the Installation

$ pig –version

Apache Pig version 0.15.0 (r1682971) compiled August 28 2023, 10:24:15 **Apache Pig - Execution**

You can run Apache Pig in two modes, namely, Local Mode and HDFS mode.

Local Mode
In this mode, all the files are installed and run from your local host and local file system. There is no need of Hadoop or HDFS. This mode is generally used for testing purpose.

MapReduce Mode
MapReduce mode is where we load or process the data that exists in the Hadoop File System (HDFS) using Apache Pig. In this mode, whenever we execute the Pig Latin statements to process the data, a MapReduce job is invoked in the back-end to perform a particular operation on the data that exists in the HDFS.

Apache Pig Execution Mechanisms
Apache Pig scripts can be executed in three ways, namely, interactive mode, batch mode, and embedded mode.

Interactive Mode (Grunt shell) − You can run Apache Pig in interactive mode using the Grunt shell. In this shell, you can enter the Pig Latin statements and get the output (using Dump operator).

Batch Mode (Script) − You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with .pig extension.

Embedded Mode (UDF) − Apache Pig provides the provision of defining our own functions (User Defined Functions) in programming languages such as Java, and using them in our script.

Invoking the Grunt Shell

You can invoke the Grunt shell in a desired mode (local/MapReduce) using the −x option as shown below.

Local Mode : **$pig –x local**
Map Reduce mode: **$pig –x mapreduce**

Either of these commands gives you the Grunt shell prompt as shown below.
Grunt Shell **grunt >**

You can exit the Grunt shell using 'ctrl + d'.

LOADING Data into Grunt Shell

Syntax: **DATA = LOAD <CLASSPATH> USING PigStorage(DELIMITER) as (ATTRIBUTE : DataType1, ATTRIBUTE : DataType2…..)**

Example: grunt> customers = LOAD 'customers.txt' USING PigStorage(',');

Executing Apache Pig in Batch Mode

You can write an entire Pig Latin script in a file and execute it using the –x command. Example: we have a Pig script in a file named sample_script.pig as shown below.

 Sample_script.pig student=LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING PigStorage(',') as (id:int,name:chararray,city:chararray);

Local mode: $ pig -x local Sample_script.pig

MapReduce mode: $ pig -x mapreduce Sample_script.pig

Step 4

Describe Data **grunt>describe**

Step 5

Dump Data **grunt>dump**

Basic Pig Commands

1.   Fs: This will list all the files in the HDFS

grunt> fs –ls

2.   Clear: This will clear the interactive Grunt shell.

grunt> clear

3.   History: This command shows the commands executed so far.
grunt> history

4.   Reading Data: Assuming the data resides in HDFS, we need to read data to Pig.

grunt> college_students = LOAD 'hdfs://localhost:9000/pig_data/college_data.txt' USING PigStorage(',') as (id: int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);

PigStorage() is the function that loads and stores data as structured text files.

5.   Storing Data: The store operator stores the processed/loaded data.

grunt> STORE college_students INTO ' hdfs://localhost:9000/pig_Output/ ' USING PigStorage (',');

Here, "/pig_Output/" is the directory where the relation needs to be stored.

6.   Dump Operator: This command displays the results on the screen. It usually helps in debugging.

grunt> Dump college_students;

7.   Describe Operator: It helps the programmer view the relation's schema.

grunt> describe college_students;

8.   Explain: This command helps to review the logical, physical, and map-reduce execution plans. **grunt> explain college_students;**

9.   Illustrate operator: This gives step-by-step execution of statements in Pig Commands.

grunt> illustrate college_students;

10.   Group: This command works towards grouping data with the same key.

grunt> group_data = GROUP college_students by first name;

11.   COGROUP: It works similarly to the group operator. The main difference between Group & Cogroup operators is that the group operator usually uses one relation, while cogroup uses more than one relation.

12.   Join: This is used to combine two or more relations.

Example: To perform self-join, the relation "customer" is loaded from HDFS tp pig commands in two relations, customers1 & customers2.

grunt> customers3 = JOIN customers1 BY id, customers2 BY id;

Join could be self-join, Inner-join, or Outer-join.

13.   Cross: This pig command calculates the cross product of two or more relations.

grunt> cross_data = CROSS customers, orders;

14.   Union: It merges two relations. The condition for merging is that the relation's columns and domains must be identical.

grunt> student = UNION student1, student2;

15.   Filter: This helps filter out the tuples out of relation based on certain conditions.

filter_data = FILTER college_students BY city == 'Chennai';

16.   Distinct: This helps in the removal of redundant tuples from the relation.

grunt> distinct_data = DISTINCT college_students;

This filtering will create a new relation name, "distinct_data."

17.   Foreach: This helps generate data transformation based on column data.

grunt> foreach_data = FOREACH student_details GENERATE id,age,city;
This will get each student's id, age, and city values from the relation student_details and store it in another relation named foreach_data.

18.   Order by: This command displays the result in a sorted order based on one or more fields.
grunt> order_by_data = ORDER college_students BY age DESC;

This will sort the relation "college_students" in descending order by age.

Practical 9

Aim:- Configure HBase and try different HBase commands.

Prerequisites
It is essential that you have Hadoop and Java installed on your system before you go for Hbase.

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 tpmasterlocalhost .localdomain. out

Installing HBase in Pseudo-Distributed Mode Configuring HBase in pseudo-distributed

mode

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

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

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 sequences 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.

2023-08-26 11:33:47,526 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 hbase(main):001:0>

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

HBase Tables

Setting Java Environment

We can also communicate with HBase using Java libraries, but before accessing HBase using Java API you need to set classpath for those libraries.

Setting the Classpath

Before proceeding with programming, set the classpath to HBase libraries in .bashrc file. Open .bashrc in any of the editors as shown below.

$ gedit ~/.bashrc

Set classpath for HBase libraries (lib folder in HBase) in it as shown below.

export CLASSPATH = $CLASSPATH://home/hadoop/hbase/lib/*

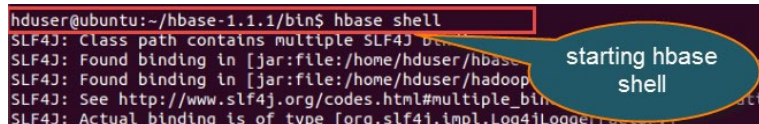This is to prevent the "class not found" exception while accessing the HBase using java API.

HBase Shell Commands

In HBase, interactive shell mode is used to interact with HBase for table operations, table management, and data modeling. By using Java API model, we can perform all type of table and data operations in HBase. We can interact with HBase using this both methods.

The only difference between these two is Java API use java code to connect with HBase and shell mode use shell commands to connect with HBase.

To get enter into HBase shell command, first of all, we have to execute the code as mentioned below:

$hbase Shell

General Commands
- **status -** This command will give details about the system status like a number of servers present in the cluster, active server count, and average load value. You can also pass any particular parameters depending on how detailed status you want to know about the system. The parameters can be
„summary", „simple", or „detailed", the **default parameter provided is "summary".**

Example: Syntax: status

hbase(main):001:0>status hbase(main):002:0>status 'simple' hbase(main):003:0>status 'summary' hbase(main):004:0> status 'detailed'

- **version -** Provides the version of HBase being used.
  Syntax: version

  Example: hbase(main):005:0> version

- **table_help** - Provides help for table-reference commands.
  Syntax: table_help

- **whoami -** It is used to return the current HBase user information from the HBase cluster.
  Syntax: Whoami

  Tables Managements commands
- **create** - Creates a table.
  Syntax: create <tablename>, <columnfamilyname> Example:- hbase(main):001:0> create 'education'
  ,'rahul' 0 rows(s) in 0.312 seconds
  =>Hbase::Table – education

- **list** - Lists all the tables in HBase.
  Syntax: list

- **describe** - Provides the description of a table. **Syntax:describe <table name>**

  Example: hbase(main):010:0>describe

  'education'

- **disable** - Disables a table. If table needs to be deleted or dropped, it has to disable first **Syntax: disable <tablename>**

  Example: hbase(main):011:0>disable
  'education'

- **is_disabled** - Verifies whether a table is disabled.

- **enable** - Enables a table.
  Syntax: enable <tablename>

  Example: hbase(main):012:0>enable 'education'

- **show_filters:** It displays all the filters present in HBase like ColumnPrefix Filter, TimestampsFilter, PageFilter, FamilyFilter, etc.
  Syntax: show_filters

- **is_enabled** - Verifies whether a table is enabled. **Syntax: is_enabled 'education'**

  This command will verify whether the named table is enabled or not. Difference between enable and is_enable command are:
- Suppose a table is disabled, to use that table we have **to enable** it by using **enable command**
- **is_enabled command** will check either the table **is enabled or not**
- **alter** - Alters a table.
  Syntax: alter <tablename>, NAME=><column familyname>, VERSIONS=>5

Example 1: To change or add the „rahul_1" column family in table „education" from current value to keep a maximum of 5 cell VERSIONS
 hbase> alter 'education', NAME='rahul_1', VERSIONS=>5

Example 2: You can also operate the alter command on several column families as well.
For example, we will define two new column to our existing table "education".

hbase> alter 'edu', 'rahul_1', {NAME => 'rahul_2', IN_MEMORY => true}, {NAME => 'rahul_3', VERSIONS => 5}

Example 3: how to delete column family from the table. To delete the „f1" column family in table
„education".

hbase> alter 'education', NAME => 'f1', METHOD => 'delete' hbase> alter 'education', 'delete' =>' rahul_1'

Example 4: how to change table scope attribute and how to remove the table scope attribute.
hbase> alter <'tablename'>, MAX_FILESIZE=>'132545224'

- **exists** - Verifies whether a table exists.

- **drop** - Drops a table from HBase.
  Syntax:drop <table name>

  Example: hbase(main):017:0>drop 'education'

We have to observe below points for drop command
- To delete the table present in HBase, first we have to disable it
- To drop the table present in HBase, first we have to disable it
- So either table to drop or delete first the table should be disable using disable command
- Here in above screenshot we are dropping table "education."
- Before execution of this command, it is necessary that you disable table "education."

- **drop_all** - Drops the tables matching the „regex" given in the command. **Syntax: drop_all<"regex">**

- **alter_status:** you can get the status of the alter command **Syntax: alter_status 'education'**

- **Java Admin API** - Prior to all the above commands, Java provides an Admin API to achieve DDL functionalities through programming. Under org.apache.hadoop.hbase.client package, HBaseAdmin and HTableDescriptor are the two important classes in this package that provide DDL functionalities.

Data Manipulation Language

- **put** - Puts a cell value at a specified column in a specified row in a particular table. **Syntax: put <'tablename'>,<'rowname'>,<'columnvalue'>,<'value'>**

  Example: hbase> put 'rahul', 'r1', 'c1', 'value', 10 hbase> g.put 'guru99', 'r1', 'c1', 'value', 10 (Command for Table Reference)

- **get** - Fetches the contents of row or a cell.
  Syntax: get <'tablename'>, <'rowname'>, {< Additional parameters>}

  Here <Additional Parameters> include TIMERANGE, TIMESTAMP, VERSIONS and FILTERS.
  Example: hbase> get 'rahul', 'r1', {COLUMN => 'c1'} (Row and Column Values Display)

  hbase> get 'rahul', 'r1'   (Only Row Values Display)

  hbase> get 'rahul', 'r1', {TIMERANGE => [ts1, ts2]} (Row 1 values in the time range ts1 and ts2 will be displayed)

  hbase> get 'rahul', 'r1', {COLUMN => ['c1', 'c2', 'c3']}(row r1 and column families" c1, c2, c3 values will be displayed)

- **delete** - Deletes a cell value in a table.
  Syntax:delete <'tablename'>,<'row name'>,<'column name'> Example: hbase(main):)020:0> delete 'guru99', 'r1', 'c1".
- **deleteall** - Deletes all the cells in a given row. **Syntax: deleteall <'tablename'>, <'rowname'> Example:** hbase>deleteall 'guru99', 'r1', 'c1'
- **scan** - Scans and returns the table data.
  Syntax: scan <'tablename'>, {Optional parameters} Example: hbase(main):)016:0> scan 'rahul'
- **count** - Counts and returns the number of rows in a table. **Syntax: count <'tablename'>, CACHE**

  =>1000 Example: hbase> count 'rahul',

  CACHE=>1000

  We can run the count command on table reference also like below **hbase>g.count INTERVAL=>100000 hbase>g.count INTERVAL=>10, CACHE=>1000**
- **truncate** - Disables, drops, and recreates a specified table. **Syntax: truncate <tablename>**

  Example: hbase(main):)027:0> truncate edu""""

- **Java client API** - Prior to all the above commands, Java provides a client API to achieve DML functionalities, CRUD (Create Retrieve Update Delete) operations and more through programming, under org.apache.hadoop.hbase.client package. HTable Put and Get are the important classes in this package.

Practical 10

Aim:- Write a java program to insert, update and delete records from HBase.

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import java.io.IOException;

public class HBaseCRUDExample { public static void main(String[] args) throws IOException {
// Create configuration
Configuration config = HBaseConfiguration.create();

// Specify HBase configuration details config.set("hbase.zookeeper.quorum", "localhost"); //
ZooKeeper quorum
config.set("hbase.zookeeper.property.clientPort", "2181"); // ZooKeeper client port

// Create HBase connection
        try (Connection connection = ConnectionFactory.createConnection(config)) {
        // Specify the table name
        TableName tableName = TableName.valueOf("your_table_name");

// Instantiate the Table instance
Table table = connection.getTable(tableName);

// Insert a record
Put put = new Put(Bytes.toBytes("row1"));
put.addColumn(Bytes.toBytes("cf"), Bytes.toBytes("col1"), Bytes.toBytes("value1"));
table.put(put);

// Retrieve the record
Get get = new Get(Bytes.toBytes("row1"));
Result result = table.get(get);
byte[] value = result.getValue(Bytes.toBytes("cf"), Bytes.toBytes("col1"));
System.out.println("Retrieved Value: " + Bytes.toString(value));

// Update the record
Put updatePut = new Put(Bytes.toBytes("row1"));
updatePut.addColumn(Bytes.toBytes("cf"),
Bytes.toBytes("col1"),Bytes.toBytes("updated_value"));
table.put(updatePut);

// Retrieve the updated record
Result updatedResult = table.get(get);
byte[] updatedValue = updatedResult.getValue(Bytes.toBytes("cf"), Bytes.toBytes("col1"));
System.out.println("Updated Value: " + Bytes.toString(updatedValue));
```

```
// Delete the record
Delete delete = new Delete(Bytes.toBytes("row1"));

table.delete(delete);

// Verify deletion
Result deletedResult = table.get(get);
if(deletedResult.isEmpty()) {
System.out.println("Record deleted successfully.");
} else {
System.out.println("Record deletion failed.");
}


// Close the table table.close();
}
}
}
```

Note:

- Before you proceed, make sure you have the HBase client libraries in your classpath.
- Remember to replace **"your_table_name"** with the actual name of the table you want to interact with. Also, make sure that your HBase server is running and properly configured.

Practical 11

Aim:- Install Apache Spark and try basic commands.

Step 1: Verifying Java Installation
$java –version

Step 2: Verifying Scala installation
You should Scala language to implement Spark.
$scala –version

If Scala is already installed on your system, you get to see the following response − **Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL**

In case you don't have Scala installed on your system, then proceed to next step for Scala

installation. Step 3: Downloading Scala

Download the latest version of Scala by visit the following link **https://www.scala-lang.org/download/.** After downloading, you will find the Scala tar file in the download folder.

Step 4: Installing Scala

Extract the Scala tar file
$ tar xvf scala-2.11.6.tgz

Move Scala software files
Use the following commands for moving the Scala software files, to respective directory **(/usr/local/scala).**

$ su –
Password
:
 # cd /home/Hadoop/Downloads/
 # mv scala-2.11.6
/usr/local/scala # exit

Set PATH for Scala
$ export PATH =

$PATH:/usr/local/scala/bin Verifying Scala

Installation

After installation, it is better to verify it.
$scala -version
Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL

Step 5: Downloading Apache Spark

Download the latest version of Spark by visiting the given link
https://spark.apache.org/downloads.html. After downloading it, you will find the Spark tar
file in the download folder.

Step 6: Installing
Spark Extracting

Spark tar
        $ tar xvf spark-1.3.1-binhadoop2.6.tgz Moving Spark software files

The following commands for moving the Spark software files to respective directory
(/usr/local/spark).
$ su –
Password
:

 # cd /home/Hadoop/Downloads/
 # mv spark-1.3.1-bin-hadoop2.6
/usr/local/spark # exit

Setting up the environment for Spark
Add the following line to ~**/.bashrc** file. It means adding the location, where the spark software
files are located to the PATH variable.

export PATH=$PATH:/usr/local/spark/bin

Use the following command for sourcing the ~/.bashrc file.

$ source ~/.bashrc

Step 7: Verifying the Spark Installation

$spark-shell

If spark is installed successfully then you will find the following output.

Spark assembly has been built with Hive, including Datanucleus jars on classpath Using
Spark's default log4j profile: org/apache/spark/log4jdefaults.properties 16/09/23 10:50:14
INFO SecurityManager: Changing view acls to: hadoop 16/09/23 10:50:14 INFO
SecurityManager: Changing modify acls to: hadoop
 16/09/23 10:50:14 INFO SecurityManager: SecurityManager: authentication disabled; ui acls
disabled; users with view permissions: Set(hadoop); users with modify
permissions: Set(hadoop)
16/09/23 10:50:15 INFO HttpServer: Starting HTTP Server
16/09/23 10:50:17 INFO Utils: Successfully started service 'HTTP class server' on port 43292.
Welcome to

 _/ __/_ _____/ / /__

```
_\ \/ _ \/ _ `/ __/ ' _/
                /__/ ._\_,_/_/ /_/\_\  version 1.4.0
/_/
```

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_71) Type in expressions to have them evaluated. Spark context available as sc scala>

Apache Spark Commands:

Apache Spark is a powerful open-source framework for big data processing and analytics. It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.

Starting Spark Shell: spark-shell

This launches the Spark shell in Scala, where you can execute Spark code interactively.

Creating RDD (Resilient Distributed Dataset):

RDD is the fundamental data structure in Spark. You can create an RDD from various sources, such as text files, HDFS, or by transforming an existing RDD.

Create a new RDD

a)   Read File from local filesystem and create an RDD. [php]scala> val data =

sc.textFile("data.txt")[/php]

Note: sc is the object of SparkContext

Note: You need to create a file data.txt in Spark_Home directory

b)   Create an RDD through Parallelized Collection

[php]scala> val no = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) scala> val noData = sc.parallelize(no)[/php]

c)   From Existing RDDs

[php]scala> val newRDD = no.map(data => (data * 2))[/php]

These are **three methods to create the RDD**. We can use the first method, when data is already available with the external systems like a local filesystem, **HDFS, HBase, Cassandra, S3**, etc. One can create an RDD by calling a **textFile** method of **Spark Context** with path / URL as the argument. The second approach can be used with the existing collections and the third one is a way to create new RDD from the existing one.

Number of Items in the RDD

Count the number of items available in the RDD. To count the items we need to call an Action:

74

[php]scala>

data.count()[/php] Filter Operation
Filter the RDD and create new RDD of items which contain word "DataFlair". To filter, we need to call transformation filter, which will return a new RDD with subset of items.
[php]scala> val DFData = data.filter(line =>

line.contains("DataFlair"))[/php] Transformation and Action together
For complex requirements, we can chain multiple operations together like filter transformation and count action together:

[php]scala> data.filter(line =>

 line.contains("DataFlair")).count()[/php] Read the first item from the RDD To read the first item from the file, you can use the following command:

[php]scala>

data.first()[/php] Read the first 5

item from the RDD
To read the first 5 item from the file, you can use the following command:

        [php]scala> data.take(5)[/php] RDD Partitions
An RDD is made up of multiple partitions, to count the number of partitions:

[php]scala> data.partitions.length[/php]

Note: Minimum no. of partitions in the RDD is 2 (by default). When we create RDD from HDFS file then a number of blocks will be equals to the number of partitions.

Cache the file
Caching is the optimization technique. Once we cache the RDD in the memory all future computation will work on the in-memory data, which saves disk seeks and improve the performance.

[php]scala> data.cache()[/php]

RDD will not be cached once you run above operation, you can visit the web UI:
http://localhost:4040/storage, it will be blank. RDDs are not explicitly cached once we run **cache(),** rather RDDs will be cached once we run the Action, which actually needs data read from the disk. Let's run some actions
[php]scala> data.count()[/php]
[php]scala> data.collect()[/php]

Read Data from HDFS file
To **read data from HDFS** file we can specify complete hdfs URL like **hdfs://IP:PORT/PATH**

[php]scala> var hFile = sc.textFile("hdfs://localhost:9000/inp")[/php]

Practical 12

Aim:- Write a scala program to process CSV, JSON and TXT File.

```scala
import org.apache.spark.sql.SparkSession

object FileProcessingExample {
def main(args: Array[String]): Unit = {
// Create a Spark session
val spark = SparkSession.builder() .appName("FileProcessingExample").master("local[*]")
 // Run locally using all available cores .getOrCreate()

// Process CSV file
val csvFilePath = "path/to/csv/file.csv"
val csvDF = spark.read.option("header", "true").csv(csvFilePath) println("CSV File Contents:")
csvDF.show()

// Process JSON file val jsonFilePath = "path/to/json/file.json"
val jsonDF = spark.read.json(jsonFilePath)
println("JSON File Contents:")
jsonDF.show()

// Process TXT file
val txtFilePath = "path/to/txt/file.txt"
val txtRDD = spark.sparkContext.textFile(txtFilePath) println("TXT File Contents:")
txtRDD.collect().foreach(println)


// Stop the Spark session spark.stop()
}
}
```

In this example:

Replace **"path/to/csv/file.csv", "path/to/json/file.json", and "path/to/txt/file.txt"** with the actual file paths of your CSV, JSON, and TXT files.

- The program uses **SparkSession** to create a Spark context.
- It reads the CSV file with headers using **spark.read.option("header", "true").csv(csvFilePath).**
- It reads the JSON file using **spark.read.json(jsonFilePath).**
- It reads the TXT file using **spark.sparkContext.textFile(txtFilePath),** which creates an RDD.
- It displays the contents of each file using **show()** or **collect().**
- Remember to adjust the code based on your specific file formats and requirements. Additionally, make sure you have the appropriate Spark dependencies and configuration set up to run the program.

Practical 13

Aim:- Write a scala program

a. To get the character at the given index within a given String. Also print the length of the string
b. To compare two strings lexicographically
c. To concatenate a given string to the end of another string
d. To exchange the first and last characters in a given string and return the new string
e. To exchange the first and last characters in a given string and return the new string

```scala
object StringOperations {
def main(args: Array[String]): Unit = {
val str1 = "Hello, World!"
val str2 = "Scala Programming"

// a. Get character at index and print string length
val index = 7
println(s"Character at index $index: ${charAt(str1, index)}")
println(s"Length of the string: ${str1.length}")

// b. Compare two strings lexicographically
val comparisonResult = compareStrings(str1, str2)
if (comparisonResult < 0)
        println("String 1 comes before String 2")
else if (comparisonResult > 0)
        println("String 2 comes before String 1")
else
        println("Both strings are equal")

// c. Concatenate a string to the end of another string
val concatenatedString = concatenateStrings(str1, str2) println(s"Concatenated string:
$concatenatedString")

// d. Exchange the first and last characters in a string
val exchangedString = exchangeFirstLast(str1)
println(s"String after exchanging first and last characters: $exchangedString") }

def charAt(str: String, index: Int): Char = {
if (index >= 0 && index < str.length)
str.charAt(index)
else
throw new IllegalArgumentException("Index out of bounds") }
```

```
def compareStrings(str1: String, str2: String): Int = { str1.compareTo(str2) }
def concatenateStrings(str1: String, str2: String): String = { str1 + str2 }
def exchangeFirstLast(str: String): String = {
if (str.length <= 1) str
else
str.last + str.substring(1, str.length - 1) + str.head }
}
```
Note:

This program defines functions to perform each of the specified tasks:
a) **charAt** function gets the character at the given index within the string and uses the length method to print the length of the string.
b) **compareStrings** function compares two strings using the **compareTo** method.
c) **concatenateStrings** function concatenates one string to the end of another.
d) **exchangeFirstLast** function exchanges the first and last characters of the string and returns the modified string.

Remember to replace the sample strings with the actual strings you want to use for testing.

Practical 14

Aim:- Capstone project

T

```
In [3]:  import tensorflow as tf
         import cv2
         import os
         import matplotlib.pyplot as plt
         import numpy as np
```
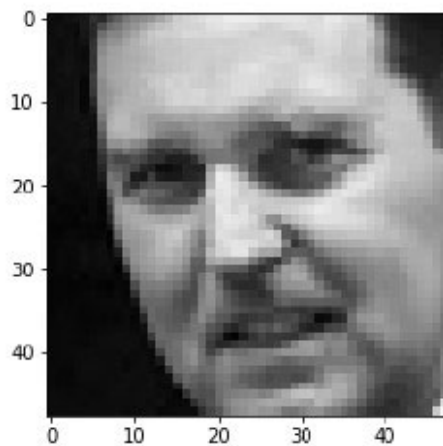
```
In [4]:  img_array=cv2.imread("Training/0/Training_99633760.jpg")
```

```
In [5]:  img_array.shape
```

```
Out[5]:  (48, 48, 3)
```

```
In [6]:  plt.imshow(img_array)
```

```
Out[6]:  <matplotlib.image.AxesImage at 0x23f185c0340>
```
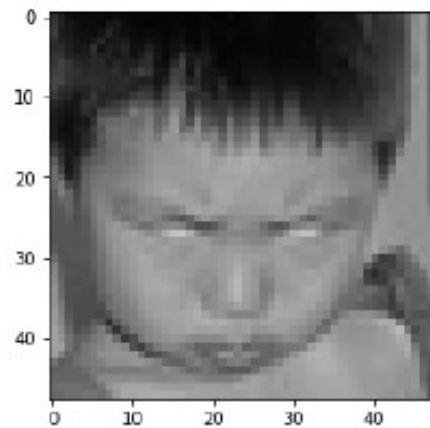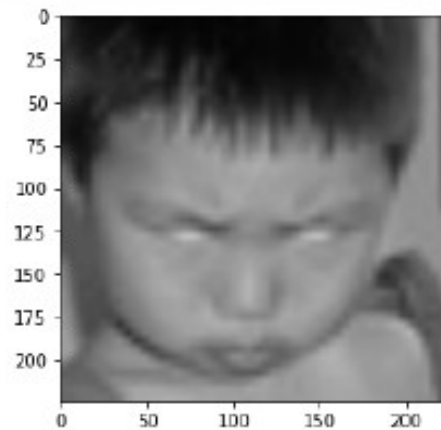


```
In [7]:  Datadirectory="Training/"
```

```
In [8]:  Classes=["0","1","2","3","4","5","6"]
```

```
In [9]:  for category in Classes:
             path=os.path.join(Datadirectory,category)
             for img in os.listdir(path):
                 img_array=cv2.imread(os.path.join(path,img))
                 plt.imshow(cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB))
                 plt.show()
                 break
             break
```

```
In [92]: img_size=224
         new_array=cv2.resize(img_array,(img_size,img_size))
         plt.imshow(cv2.cvtColor(new_array,cv2.COLOR_BGR2RGB))
         plt.show()
```



```
In [11]: new_array.shape
```

```
Out[11]: (224, 224, 3)
```

read all the images and converting them to array

```
In [12]: training_Data=[]

         def create_training_Data():
             for category in Classes:
                 count = 0
                 path=os.path.join(Datadirectory, category)
                 class_num=Classes.index(category)
                 for img in os.listdir(path):
                     try:
                         img_array=cv2.imread(os.path.join(path,img))
                         new_array=cv2.resize(img_array,(img_size,img_size))
                         training_Data.append([new_array,class_num])
```

```
        except Exception as e:
            pass
```

In [13]: `create_training_Data()`

In [14]: `print(len(training_Data))`

28709

In [15]: `temp = np.array(training_Data)`

C:\Users\dhava\AppData\Local\Temp\ipykernel_11068\2755283514.py:1: VisibleDeprecat
ionWarning: Creating an ndarray from ragged nested sequences (which is a list-or-t
uple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecate
d. If you meant to do this, you must specify 'dtype=object' when creating the ndar
ray.
  temp = np.array(training_Data)

In [16]: `temp.shape`

Out[16]: (28709, 2)

using training and deploying for real time webcam demo

In [17]:
```python
import random
random.shuffle(training_Data)
```

In [18]:
```python
X=[]
y=[]
for features,label in training_Data:
    X.append(features)
    y.append(label)
x=np.array(X).reshape(-1,img_size,img_size,3)
```

In [19]: `x.shape`

Out[19]: (28709, 224, 224, 3)

In [20]: `x=x/255.0`

In [21]: `y[1098]`

Out[21]: 0

In [22]: `Y=np.array(y)`

In [23]: `Y.shape`

Out[23]: (28709,)

deep learning model for training–Transfer Learning

```
In [24]: import tensorflow as tf
         from tensorflow import keras
         from tensorflow.keras import layers
```

```
In [25]: model = tf.keras.applications.MobileNetV2()   ## Pre trained model
```

```
In [26]: model.summary()
```

Output:-