Dr. AKHILESH DAS GUPTA INSTITUTE OF PROFESSIONAL STUDIES

(Formerly Northern India Engineering College)
FC-26, Shastri Park, New Delhi-110053

BDA MANUAL

Academic Year: 2023-2024 Semester: 6th

Name of Faculty: Dr. Archana Kumar

Department: AI&DS

Program: B.Tech Sem: 6th

Subject Code: AIDS 306P

Subject Name: BIG DATA ANALYTICS LAB

VISION

To produce globally competent and socially responsible technocrats and entrepreneurs who can develop innovative solutions to meet the challenges of 21st century.

SECTION A(GGSIPU)

INDEX

List of Experiments

S.NO.	EXPERIMENT	COURSE OUTCOMES
1	Install Apache Hadoop.	CO1
2	Develop a map reduce program to calculate the frequency of a given word in a given file.	CO1
3	Develop a map reduce program to find the maximum temperature in each year.	CO2
4	Develop a map reduce program to find the grade of students.	CO1
5	Develop a map reduce program to implement matrix multiplication.	CO1
6	Develop a map reduce program to find the maximum electrical consumption in each year given electrical consumption for each month in each year.	CO2
7	Develop a map reduce program to analyze weather data set and print whether the day is shiny or cool day.	CO2
8	Develop a map reduce program to find the tags associated with each movie by analyzing movie lens data.	CO2
9	Develop a map reduce program to analyze Uber data set to find the days on which each basement has more trips using the following data set. The uber data set consists of four columns they are: Dispatching, base, no. date active, vehicle trips.	CO2
10	Develop a map reduce program to analyze titanic dataset to find the average age of the people (both male and female) who died in the tragedy. How many people survived in each class.	CO2
11	Develop a program to calculate the maximum recorded temperature year wise for the weather data set in Pig Latin.	CO2
12	Write queries to sort and aggregate the data in a table using HiveQL.	CO2

EXPERIMENT 1

AIM: To Install Apache Hadoop.

Hadoop software can be installed in three modes of

Hadoop is a Java-based programming framework that supports the processing and storage of extremely large datasets on a cluster of inexpensive machines. It was the first major open source project in the big data playing field and is sponsored by the Apache Software Foundation.

Hadoop-2.7.3 is comprised of four main layers:

- **Hadoop Common** is the collection of utilities and libraries that support other Hadoopmodules.
- **HDFS**, which stands for Hadoop Distributed File System, is responsible for persistingdata to disk.
- **YARN**, short for Yet Another Resource Negotiator, is the "operating system" for HDFS.
- **MapReduce** is the original processing model for Hadoop clusters. It distributes work within the cluster or map, then organizes and reduces the results from the nodes into a response to a query. Many other processing models are available for the 2.x version of Hadoop.

Hadoop clusters are relatively complex to set up, so the project includes a stand-alone mode which is suitable for learning about Hadoop, performing simple operations, and debugging.

Procedure:

we'll install Hadoop in stand-alone mode and run one of the example example MapReduce programs it includes to verify the installation.

Prerequisites:

Step1: Installing Java 8

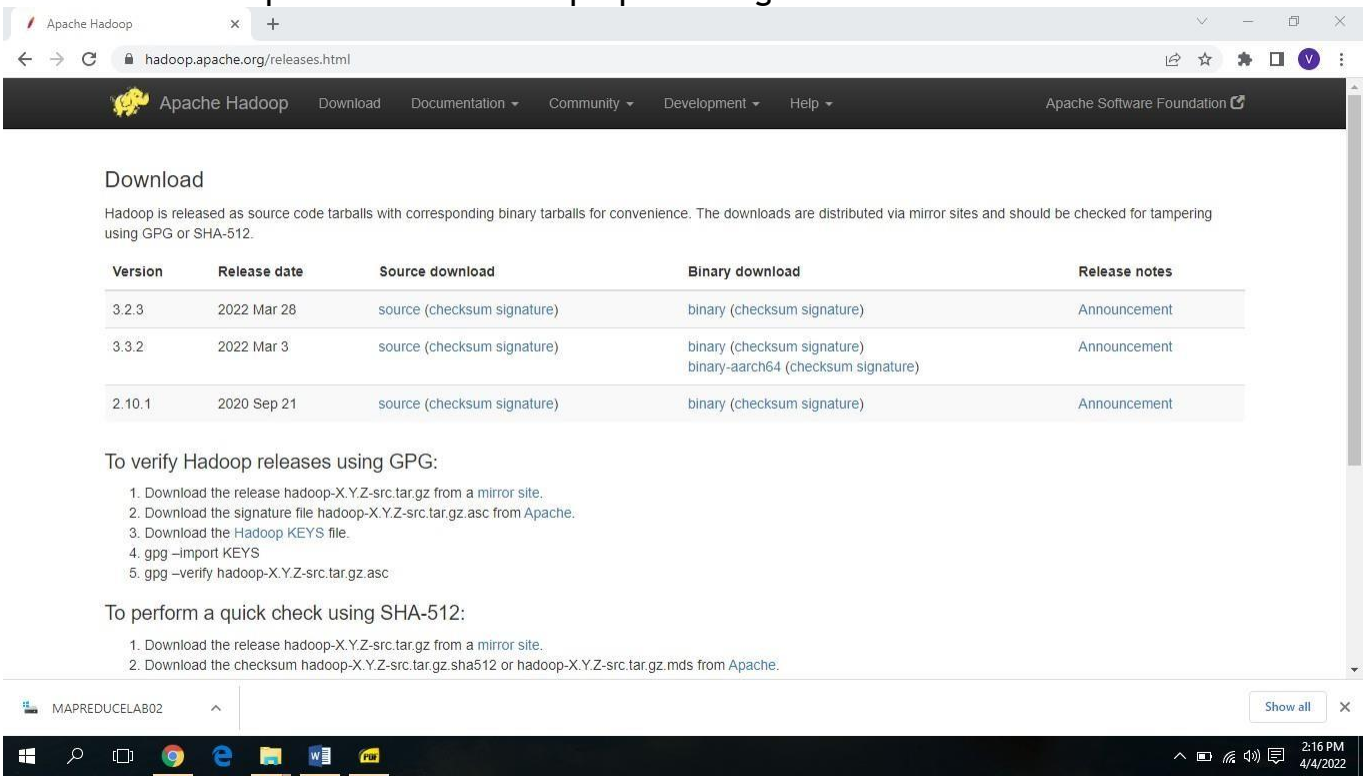
version.Openjdk version
"1.8.0_91"
OpenJDK Runtime Environment (build 1.8.0_91-8u91-b14-3ubuntu1~16.04.1-b14)
OpenJDK 64-Bit Server VM (build 25.91-b14, mixed mode)
This output verifies that OpenJDK has been successfully installed.

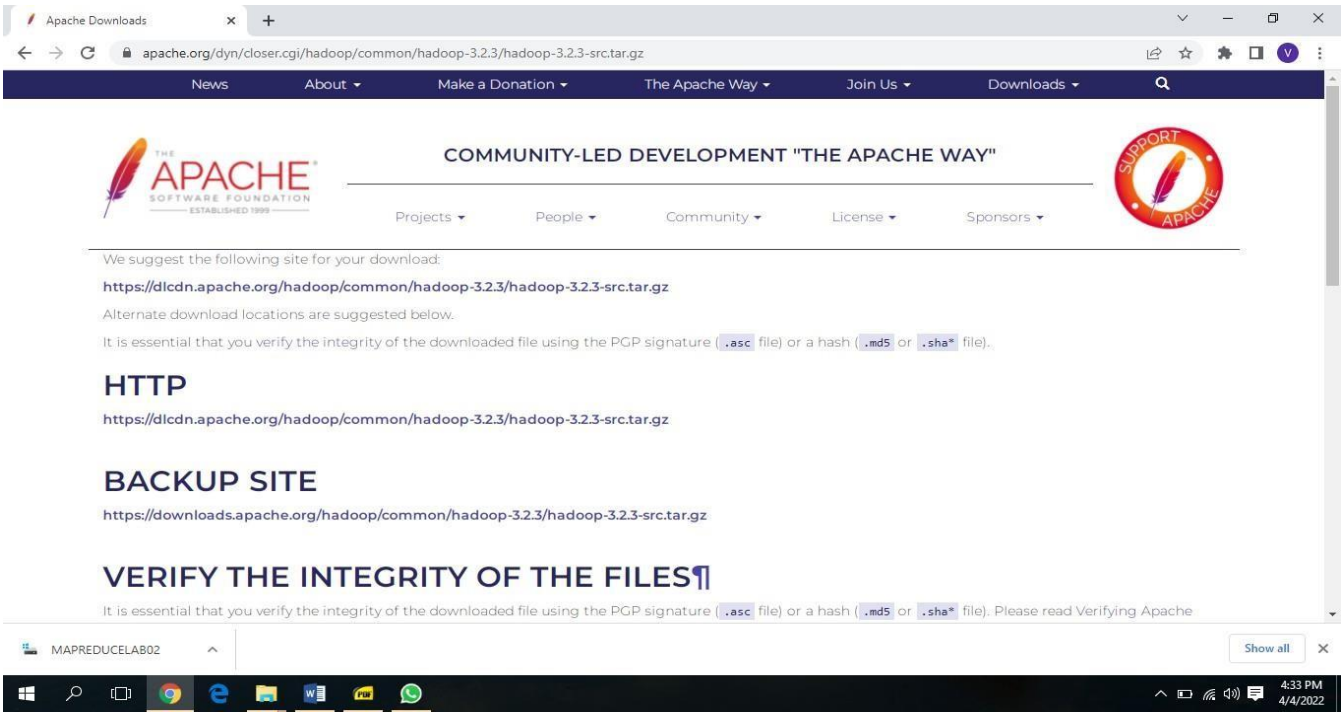
Note: To set the path for environment variables. i.e. JAVA_HOME

Step2: Installing Hadoop

With Java in place, we'll visit the Apache Hadoop Releases page to find the mostrecent stable release. Follow the binary for the current release:

Download Hadoop from www.hadoop.apache.org





Procedure to Run Hadoop

NAME - PREETI ENROLLMENT NO. - 04215611922 SECTION – T11

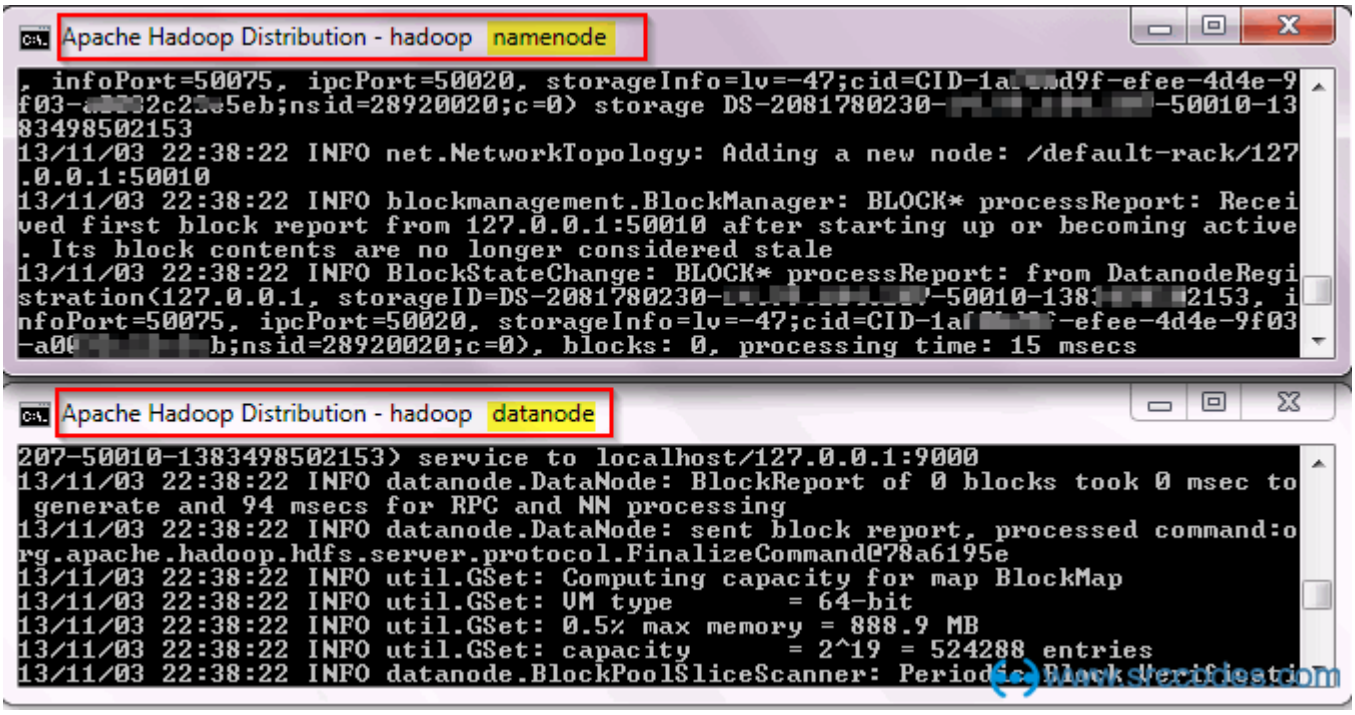
1. Install Apache Hadoop 2.2.0 in Microsoft Windows OS

If Apache Hadoop 2.2.0 is not already installed then follow the post Build, Install,Configure and Run Apache Hadoop 2.2.0 in Microsoft Windows OS.

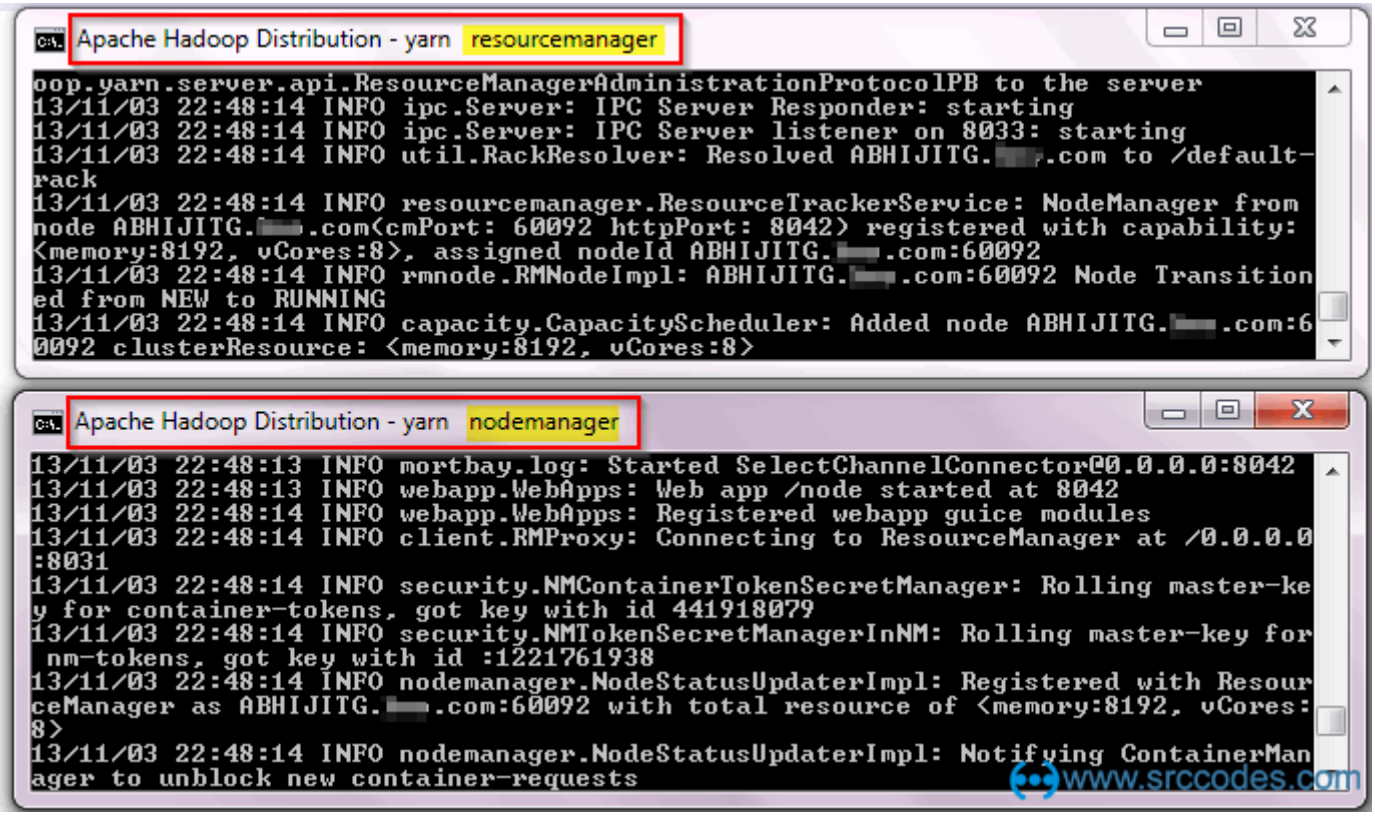
2. Start HDFS (Namenode and Datanode) and YARN (Resource Manager and NodeManager)

Run following commands.
Command Prompt
C:\Users\abhijitg>cd
c:\hadoop
c:\hadoop>sbin\start-dfs
c:\hadoop>sbin\start-yarn
starting yarn daemons

Namenode, **Datanode**, **Resource Manager** and **Node Manager** will be started infew minutes and ready to execute Hadoop **MapReduce** job in the Single Node (pseudo-distributed mode) cluster.



Resource Manager & Node Manager:



Run wordcount MapReduce job

Now we'll run **wordcount** MapReduce job available in **%HADOOP_HOME%\share\hadoop\mapreduce\hadoop-mapreduce-examples-2.2.0.jar**

Create a text file with some content. We'll pass this file as input to the **wordcount** MapReduce job for counting words.
C:\file1.txt

```
Install Hadoop

Run Hadoop Wordcount Mapreduce Example
```

Create a directory (say 'input') in HDFS to keep all the text files (say 'file1.txt') to be used for counting words.

C:\Users\abhijitg>cd c:\hadoop
C:\hadoop>bin\hdfs dfs -mkdir input

Copy the text file(say 'file1.txt') from local disk to the newly created 'input' directory in HDFS.

C:\hadoop>bin\hdfs dfs -copyFromLocal c:/file1.txt input

Check content of the copied file.

```
C:\hadoop>hdfs dfs -ls
inputFound 1 items
-rw-r--r-- 1 ABHIJITG supergroup        55 2014-02-03 13:19 input/file1.txt

C:\hadoop>bin\hdfs dfs -cat
input/file1.txtInstall Hadoop
Run Hadoop Wordcount Mapreduce Example
```

Run the wordcount MapReduce job provided in %HADOOP_HOME%\share\hadoop\mapreduce\hadoop-mapreduce-examples-2.2.0.jar

```
C:\hadoop>bin\yarn jar share/hadoop/mapreduce/hadoop-mapreduce-
examples-2.2.0.jar wordcount input output
14/02/03 13:22:02 INFO client.RMProxy: Connecting to ResourceManager at
/0.0.0.0:8032
14/02/03 13:22:03 INFO input.FileInputFormat: Total input paths to
process : 114/02/03 13:22:03 INFO mapreduce.JobSubmitter: number of
splits:1
:
:
14/02/03 13:22:04 INFO mapreduce.JobSubmitter: Submitting tokens for
job:job_1391412385921_0002
14/02/03 13:22:04 INFO impl.YarnClientImpl: Submitted
application application_1391412385921_0002 to ResourceManager
at /0.0.0.0:803214/02/03 13:22:04 INFO mapreduce.Job: The url
to track the job:
http://ABHIJITG:8088/proxy/application_1391412385921_0002/
14/02/03 13:22:04 INFO mapreduce.Job: Running job:
job_1391412385921_0002 14/02/03 13:22:14 INFO mapreduce.Job: Job
job_1391412385921_0002 running inuber mode : false
14/02/03 13:22:14 INFO mapreduce.Job:  map 0% reduce 0%
14/02/03 13:22:22 INFO mapreduce.Job:  map 100% reduce 0%
14/02/03 13:22:30 INFO mapreduce.Job:  map 100% reduce 100%
14/02/03 13:22:30 INFO mapreduce.Job: Job job_1391412385921_0002
completedsuccessfully
14/02/03 13:22:31 INFO mapreduce.Job:
      Counters: 43File System Counters
            FILE: Number of bytes read=89
            FILE: Number of bytes
            written=160142 FILE: Number of
            read operations=0 FILE: Number of
```

large read operations=0FILE: Number

of write operations=0

HDFS: Number of bytes read=171
HDFS: Number of bytes
written=59 HDFS: Number of
read operations=6
HDFS: Number of large read
operations=0HDFS: Number of write
operations=2

Job Counters
Launched map tasks=1
Launched reduce
tasks=1Data-local map
tasks=1
Total time spent by all maps in occupied slots
(ms)=5657 Total time spent by all reduces in
occupied slots (ms)=6128

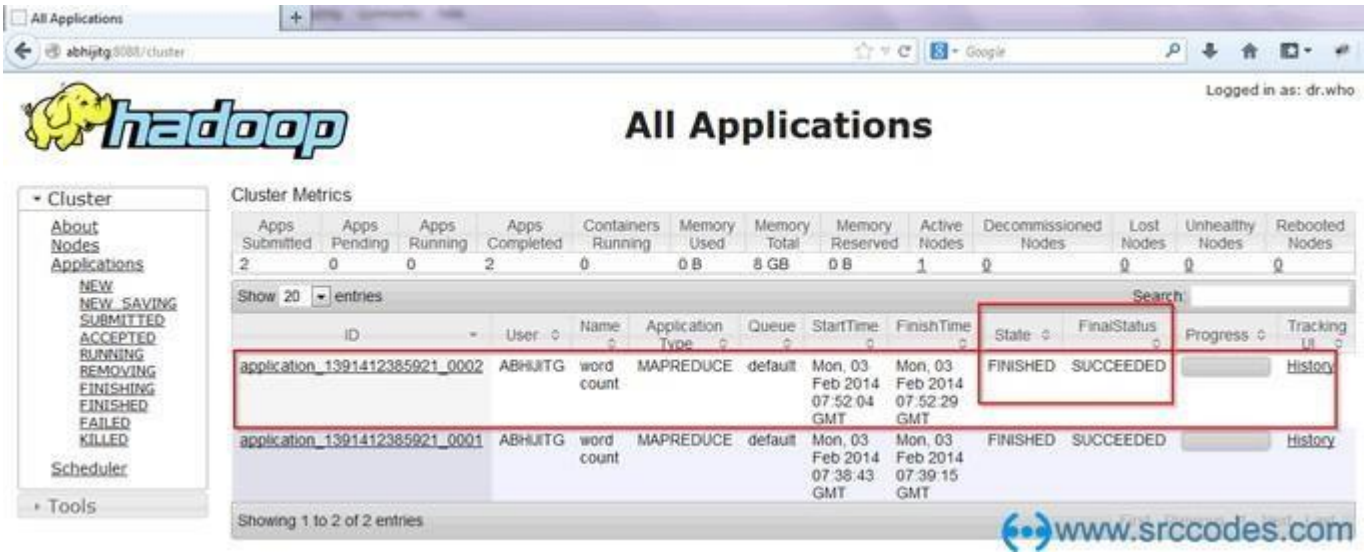
Map-Reduce Framework
Map input
records=2 Map
output records=7
Map output
bytes=82
Map output materialized
bytes=89Input split bytes=116
Combine input
records=7 Combine
output records=6
Reduce input groups=6
Reduce shuffle
bytes=89 Reduce input
records=6 Reduce
output records=6
Spilled Records=12
Shuffled Maps =1
Failed Shuffles=0
Merged Map
outputs=1
GC time elapsed
(ms)=145CPU time spent
(ms)=1418
Physical memory (bytes)
snapshot=368246784 Virtual memory (bytes)
snapshot=513716224 Total committed heap
usage (bytes)=307757056

Shuffle Errors

BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format
CountersBytes
Read=55
File Output Format Counters

Bytes Written=59

http://abhijitg:8088/cluster



Result: We've installed Hadoop in stand-alone mode and verified it by running anexample program it provided.

EXPERIMENT 2

AIM: To Develop a MapReduce program to calculate the frequency of a given word in a given file

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 tuplesinto a smaller set of tuples.

Example – (Reduce function in Word Count)

Input Set of Tuples

(output of Map

function)

(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 Program

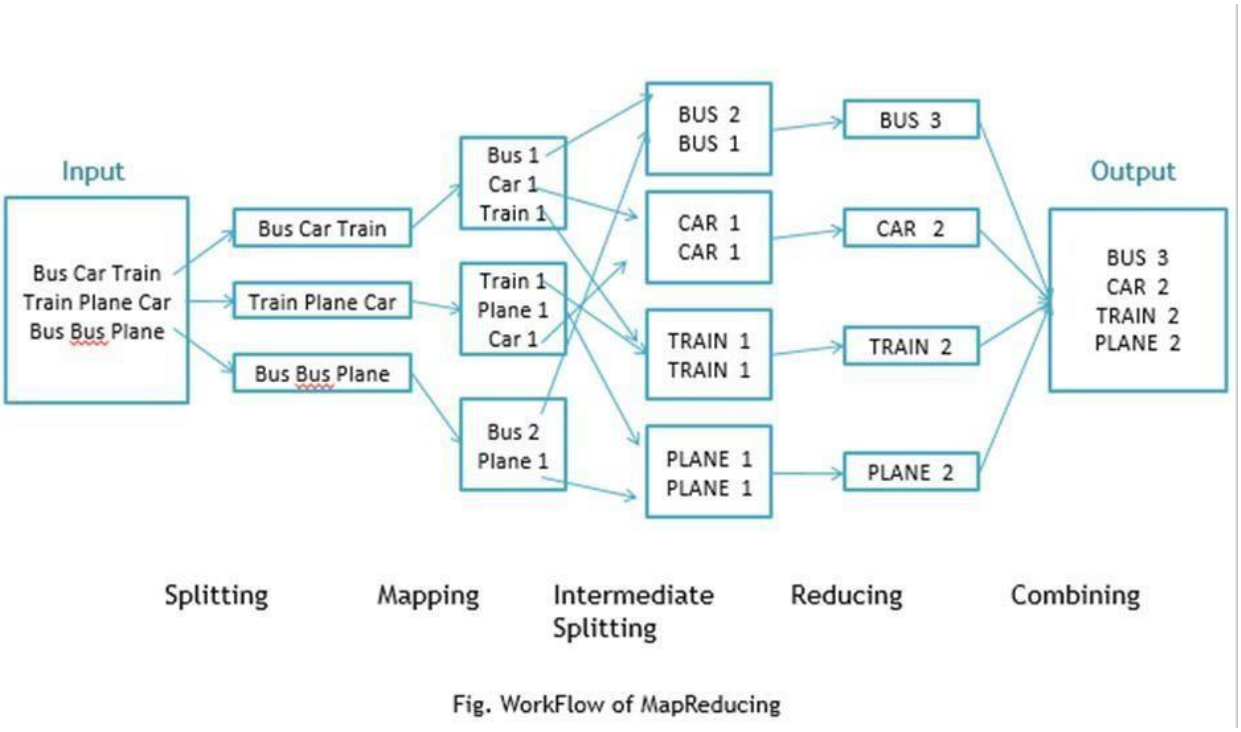


Fig. WorkFlow of MapReducing

Workflow of MapReduce consists of 5 steps

1. **Splitting** – The splitting parameter can be anything, e.g. splitting by space,comma, semicolon, or even by a new line ('\n').
2. **Mapping** – as explained above
3. Intermediate splitting – the entire process in parallel on different clusters. In orderto group them in "Reduce Phase" the similar KEY data should be on same cluster.
4. **Reduce** – it is nothing but mostly group by phase
5. **Combining** – The last phase where all the data (individual result set from eachcluster) is combine together to form a Result

Now Let's See the Word Count Program in Java

Make sure that Hadoop is installed on your system with java

idkSteps to follow

Step 1. Open Eclipse> File > New > Java Project > (Name it – MRProgramsDemo) >Finish
Step 2. Right Click > New > Package (Name it - PackageDemo) > FinishStep 3. Right Click on Package > New > Class (Name it - WordCount) Step 4. Add Following Reference Libraries –

Right Click on Project > Build Path> Add External Archivals

- /usr/lib/hadoop-0.20/hadoop-core.jar
- Usr/lib/hadoop-0.20/lib/Commons-cli-1.2.jar

Program: Step 5. Type following Program :

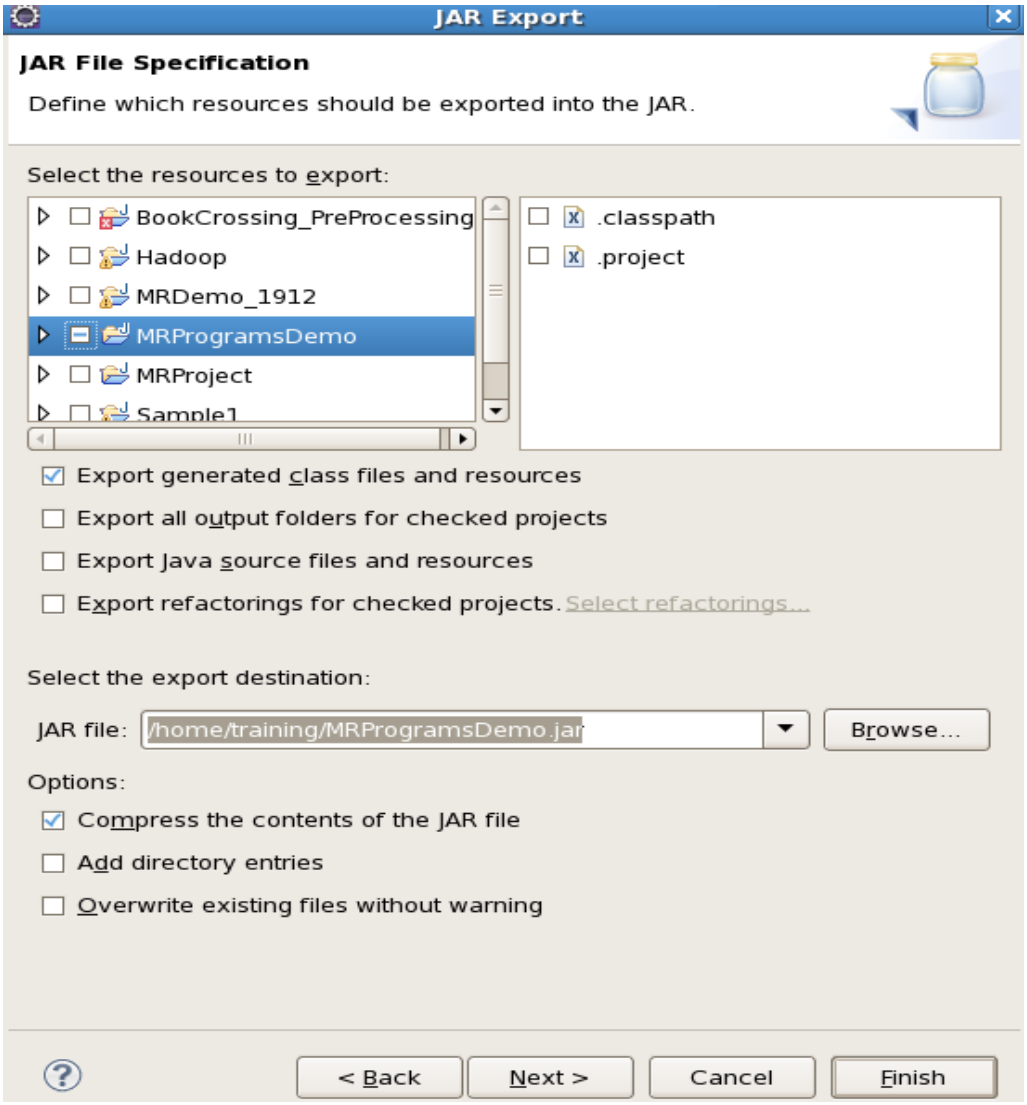
```
package PackageDemo;
import
java.io.IOException;
import
org.apache.hadoop.conf.Configuration;import
org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import
org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import
org.apache.hadoop.mapreduce.Job;
import
org.apache.hadoop.mapreduce.Mapper;
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.clas
s);
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);
```

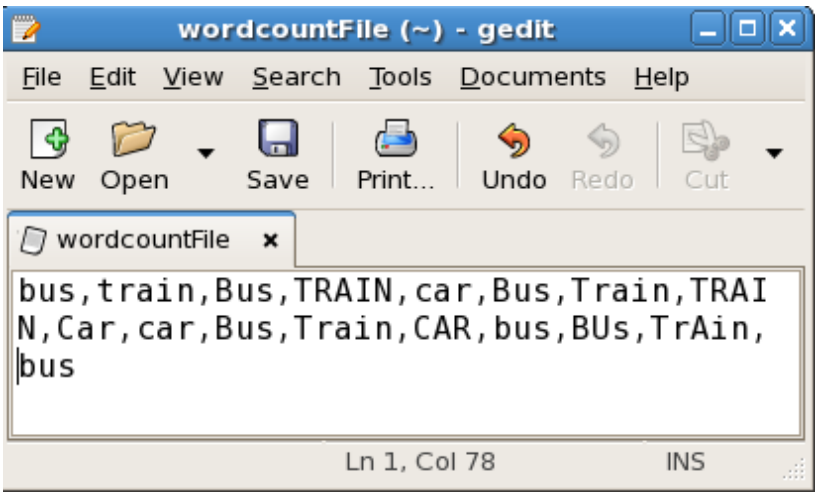
```
}
```

```
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)
throwsIOException,
InterruptedException
{
int sum = 0;
for(IntWritable value : values)
{
sum += value.get();
}
con.write(word, new IntWritable(sum));
}
}
}
```

Make Jar File

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


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
BookCrossing_PreProcessing
Hadoop
MRDemo_1912
MRProgramsDemo
MRProject
Sample1
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: /home/training/MRProgramsDemo.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Cancel
Finish


wordcountFile (~) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut
wordcountFile
bus,train,Bus,TRAIN,car,Bus,Train,TRAI
N,Car,car,Bus,Train,CAR,bus,BUs,TrAin,
bus
Ln 1, Col 78
INS

To Move this into Hadoop directly, open the terminal and enter the followingcommands:
[training@localhost ~]$ hadoop fs -put wordcountFile wordCountFile

Run Jar file

(Hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFilePathToOutputDirectry)

[training@localhost ~]$ Hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile MRDir1

Result: Open Result

[training@localhost ~]$ hadoop fs -ls
MRDir1Found 3 items
-rw-r--r-- 1 training supergroup
0 2016-02-23 03:36
/user/training/MRDir1/_SUCCESSdrwxr-xr-x
- training supergroup
0 2016-02-23 03:36 /user/training/MRDir1/_logs
-rw-r--r-- 1 training supergroup
20 2016-02-23 03:36
/user/training/MRDir1/part-r-00000
[training@localhost ~]$ hadoop fs -cat
MRDir1/part-r-00000BUS 7
CAR 4
TRAIN 6

EXPERIMENT 3

AIM: To Develop a MapReduce program to find the maximum temperature in each year.

Description: MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks.Our previous traversal has given an introduction about MapReduce This traversal explains how to design a MapReduce program. The aim of the program is to find the Maximum temperature recorded for each year of NCDC data. The input for our program is weather data files for each year This weather data is collected by National Climatic Data Center – NCDC from weather sensors at all over the world. You can find weather data for each year from ftp://ftp.ncdc.noaa.gov/pub/data/noaa/.All files are zipped by year and the weather station. For each year, there are multiple files for different weather stations.
Here is an example for 1990 (ftp://ftp.ncdc.noaa.gov/pub/data/noaa/1901/).

- 010080-99999-1990.gz
- 010100-99999-1990.gz
- 010150-99999-1990.gz
- ……

MapReduce is based on set of key value pairs. So first we have to decide on the types for the key/value pairs for the input.
Map Phase: The input for Map phase is set of weather data files as shown in snap shot. The types of input key value pairs are LongWritable and Text and the types of output key value pairs are Text and IntWritable. Each Map task extracts the temperature data from the given year file. The output of the map phase is set of key value pairs. Set of keys are the years. Values are the temperature of each year.
Reduce Phase: Reduce phase takes all the values associated with a particular key. That is all the temperature values belong to a particular year is fed to a same reducer. Then each reducer finds the highest recorded temperature for each year. The types of output key value pairs in Map phase is same for the types of input key value pairs in reduce phase (Text and IntWritable). The types of output key value pairs in reduce phase is too Text and IntWritable.
So, in this example we write three java classes:

- HighestMapper.java
- HighestReducer.java
- HighestDriver.java

Program: HighestMapper.java

```
import
java.io.IOException;
import
org.apache.hadoop.io.
*;
import org.apache.hadoop.mapred.*;
public class HighestMapper extends MapReduceBase implements
Mapper<LongWritable, Text,Text, IntWritable>
{
        public static final int MISSING = 9999;
public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output,Reporter reporter) throws IOException
{
String line =
value.toString(); String
year =
line.substring(15,19);int
temperature;
if (line.charAt(87)=='+')
temperature =
Integer.parseInt(line.substring(88, 92));else
temperature =
Integer.parseInt(line.substring(87, 92));
String quality = line.substring(92, 93);
if(temperature != MISSING &&
quality.matches("[01459]"))output.collect(new
Text(year),new IntWritable(temperature));
}
}
```

HighestReducer.java

```
import
java.io.IOException;
import
java.util.Iterator;
import
org.apache.hadoop.io.
*;
import org.apache.hadoop.mapred.*;
public class HighestReducer extends MapReduceBase implements Reducer<Text,
IntWritable,Text, IntWritable>
{
public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable>output, Reporter reporter) throws IOException
{
int max_temp = 0;
```

```
;
while (values.hasNext())
{

int
current=values.next().
get();if ( max_temp <
current) max_temp =
current;
}
output.collect(key, new IntWritable(max_temp/10));
}
```

HighestDriver.java

```
import
org.apache.hadoop.fs.Path;
import
org.apache.hadoop.conf.*;
import
org.apache.hadoop.io.*;
import
org.apache.hadoop.mapred.
*;import
org.apache.hadoop.util.*;
public class HighestDriver extends Configured
implements Tool{public int run(String[] args) throws
Exception
{
JobConf conf = new JobConf(getConf(),
HighestDriver.class);
conf.setJobName("HighestDriver");
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(HighestMapper.class);
conf.setReducerClass(HighestReducer.class);
Path inp = new
Path(args[0]);Path
out = new
Path(args[1]);
FileInputFormat.addInputPath(conf
, inp);
FileOutputFormat.setOutputPath(c
onf, out);JobClient.runJob(conf);
return 0;
}
public static void main(String[] args) throws Exception
{
int res = ToolRunner.run(new Configuration(), new
HighestDriver(),args);System.exit(res);
```

```
}
}
```

EXPERIMENT 4

AIM: To Develop a MapReduce program to find the grades of student's.

```
import
java.util.Scanner
; public class
JavaExample
{
   public static void main(String args[])
   {
         /* This program assumes that the student has 6 subjects,
          * thats why I have created the array of size 6. You can
          * change this as per the requirement.
          */
      int marks[] =
      new int[6];int i;
      float total=0, avg;
      Scanner scanner = new
      Scanner(System.in);for(i=0; i<6;
      i++) {
        System.out.print("Enter Marks of
        Subject"+(i+1)+":");marks[i] = scanner.nextInt();
        total = total + marks[i];
      }
      scanner.close();
      //Calculating
      averagehere
      avg =
      total/6;
      System.out.print("The student Grade is: ");
      if(avg>=80)
      {
         System.out.print("A");
      }
      else if(avg>=60 && avg<80)
      {
        System.out.print("B");
      }
```

```
        else if(avg>=40 && avg<60)
        {


           System.out.print("C");
        }
            else
        {

           System.out.print("D");

        }
    }
}
```

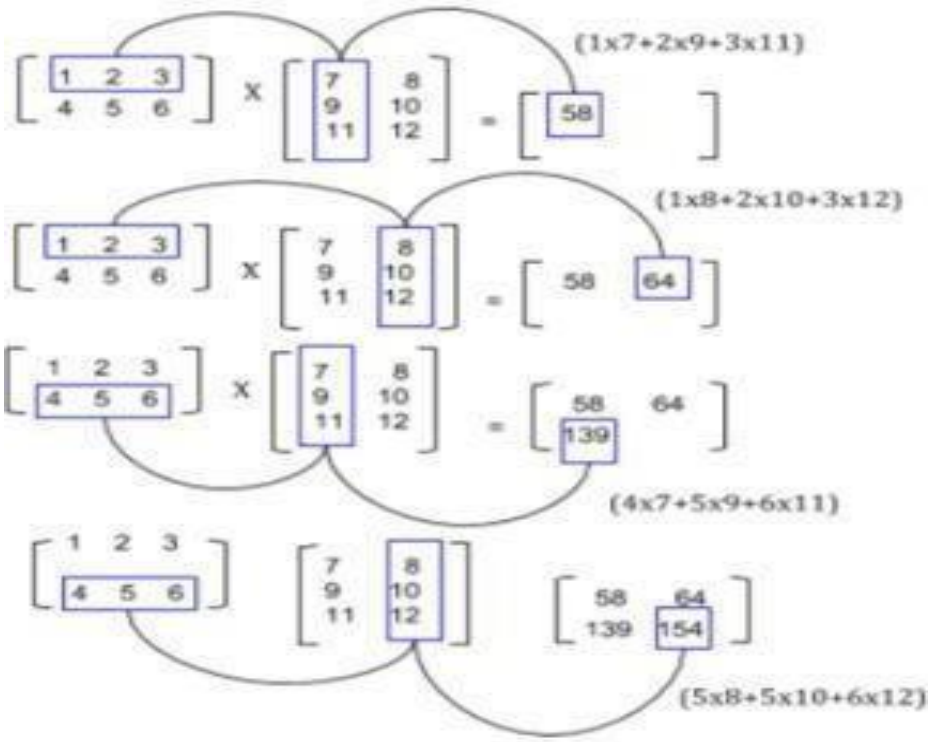
Expected Output:

```
Enter Marks of Subject1:40
Enter Marks of Subject2:80
Enter Marks of Subject3:80
Enter Marks of Subject4:40
Enter Marks of Subject5:60 Enter Marks of Subject6:60
The student Grade is: B
```

EXPERIMENT 5

AIM: To Develop a Map Reduce program to implement Matrix Multiplication.

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



Algorithm for Map Function.

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

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

number ofcolumns of N

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

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

Algorithm for Reduce Function.

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

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

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

Step 2. Creating Mapper file for Matrix Multiplication.

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

import
org.apache.hadoop.conf.Configuratio
n;import org.apache.hadoop.fs.Path;
import
org.apache.hadoop.io.DoubleWritabl
e;import
org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import
```

```
org.apache.hadoop.io.WritableComparabl
e;import
org.apache.hadoop.mapreduce.Job;

import
org.apache.hadoop.mapreduce.Mapper;
import
org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.mapreduce.lib.input.
*; import
org.apache.hadoop.mapreduce.lib.output
.*;import
org.apache.hadoop.util.ReflectionUtils;

class Element implements
        Writable {int tag;
        int
        index
        ;
        doubl
        e
        value
        ;
        Elem
        ent()
        {
                tag = 0;
                index = 0;
                value = 0.0;
        }
        Element(int tag, int index,
                double value) {this.tag =
                tag;
                this.index
                = index;
                this.value
                = value;
        }
        @Override
        public void readFields(DataInput input) throws
                IOException {tag = input.readInt();
                index =
                input.readInt();
                value =
                input.readDouble();
        }
        @Override
```

```
        public void write(DataOutput output) throws
                IOException {output.writeInt(tag);
                output.writeInt(index
                );
                output.writeDouble(v
                alue);
        }
}
class Pair implements
        WritableComparable<Pair> {int i;
        int j;

        Pair() {
                i = 0;

                j = 0;
        }
        Pair(int i, int j) {
                t
                h
                is
                .i
                =
                i;
                t
                h
                is
                .j
                =
                j;
        }
        @Override
        public void readFields(DataInput input) throws
                IOException {i = input.readInt();
                j = input.readInt();
        }
        @Override
        public void write(DataOutput output) throws
                IOException {output.writeInt(i);
                output.writeInt(j);
        }
        @Override
        public int compareTo(Pair
                compare) {if (i >
                compare.i) {
                        return 1;
                } else if ( i <
                        compare.i) {
```

```
                return -1;
            } else {
                if(j > compare.j) {
                    return 1;
                } else if (j <
                    compare.
                    j) {return
                    -1;
                }
            }
            return 0;
        }
        public String
            toString() {
            return i + " " +
            j + " ";
        }
    }
    public class Multiply {
        public static class MatriceMapperM extends Mapper<Object,Text,IntWritable,Element>
    {



            @Override
            public void map(Object key, Text value, Context
                    context) throws IOException,
                    InterruptedException {
                String readLine =
                value.toString(); String[]
                stringTokens = readLine.split(",");

                int index = Integer.parseInt(stringTokens[0]);
                double elementValue =
                Double.parseDouble(stringTokens[2]);Element e =
                new Element(0, index, elementValue); IntWritable
                keyValue = new
    IntWritable(Integer.parseInt(stringTokens[1]));
                context.write(keyValue, e);
            }
        }
        public static class MatriceMapperN extends
            Mapper<Object,Text,IntWritable,Element> {@Override
            public void map(Object key, Text value, Context
                    context) throws IOException,
                    InterruptedException {
                String readLine =
                value.toString(); String[]
                stringTokens = readLine.split(",");
```

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

                ReflectionUtils.copy(conf, element, tempElement);

                if (tempElement.tag
                    == 0) {
                    M.add(tempEl
                    ement);
                } else
                    if(tempElement.t
                    ag == 1) {
                    N.add(tempEleme
                    nt);
                }
            }
            for(int i=0;i<M.size();i++) {
                for(int j=0;j<N.size();j++) {

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

                    context.write(p, new DoubleWritable(multiplyOutput));
                    }
                }
            }
        }
```

```
        public static class MapMxN extends Mapper<Object, Text, Pair,
                DoubleWritable> {@Override
                public void map(Object key, Text value, Context
                                context) throws IOException,
                                InterruptedException {
                        String readLine =
                        value.toString(); String[]
                        pairValue = readLine.split(" ");
                        Pair p = new
Pair(Integer.parseInt(pairValue[0]),Integer.parseInt(pairValue[1]));
                        DoubleWritable val = new
DoubleWritable(Double.parseDouble(pairValue[2]));
                        context.write(p, val);
                }
        }
        public static class ReduceMxN extends Reducer<Pair,
DoubleWritable, Pair,DoubleWritable> {
                @Override
                public void reduce(Pair key, Iterable<DoubleWritable> values, Context
context)throws IOException, InterruptedException {
                        double sum = 0.0;
                        for(DoubleWritable value :
                        values) {

                                sum += value.get();
                        }
                        context.write(key, new DoubleWritable(sum));
                }
        }
        public static void main(String[] args) throws
                Exception {Job job = Job.getInstance();
                job.setJobName("MapIntermediate");
                job.setJarByClass(Project1.class);
                MultipleInputs.addInputPath(job, new Path(args[0]),
TextInputFormat.class,MatriceMapperM.class);
                MultipleInputs.addInputPath(job, new Path(args[1]),
TextInputFormat.class,MatriceMapperN.class);
                job.setReducerClass(ReducerMxN.class);
                job.setMapOutputKeyClass(IntWritable.class);
                job.setMapOutputValueClass(Element.class);
                job.setOutputKeyClass(Pair.class);
                job.setOutputValueClass(DoubleWritable.class);
                job.setOutputFormatClass(TextOutputFormat.cl
                ass); FileOutputFormat.setOutputPath(job, new
                Path(args[2]));job.waitForCompletion(true);
                Job job2 = Job.getInstance();
                job2.setJobName("MapFinalOutput");
                job2.setJarByClass(Project1.class);
```

```
                job2.setMapperClass(MapMxN.class);
                job2.setReducerClass(ReduceMxN.class);

                job2.setMapOutputKeyClass(Pair.class);
                job2.setMapOutputValueClass(DoubleWritable.class);

                job2.setOutputKeyClass(Pair.class);
                job2.setOutputValueClass(DoubleWritable.class);

                job2.setInputFormatClass(TextInputFormat.class);
                job2.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.setInputPaths(job2, new
                Path(args[2]));
                FileOutputFormat.setOutputPath(job2, new
                Path(args[3]));




                job2.waitForCompletion(true);
        }
}
```

Step 5. Compiling the program in particular folder named as operation

```
#!/bin/bash

rm -rf multiply.jar classes

module load

hadoop/2.6.0

mkdir -p classes
javac -d classes -cp classes:`$HADOOP_HOME/bin/hadoop classpath`
Multiply.javajar cf multiply.jar -C classes .

echo "end"
```

Step 6. Running the program in particular folder named as operation

```
export
HADOOP_CONF_DIR=/home/$USER/cometcluster
module load hadoop/2.6.0
myhadoop-
```

configure.sh
start-dfs.sh
start-yarn.sh

hdfs dfs -mkdir -p /user/$USER
hdfs dfs -put M-matrix-large.txt /user/$USER/M-matrix-large.txthdfs dfs -put N-matrix-large.txt /user/$USER/N-matrix-large.txt
hadoop jar multiply.jar edu.uta.cse6331.Multiply /user/$USER/M-matrix-large.txt /user/$USER/N-matrix-large.txt /user/$USER/intermediate
/user/$USER/outputrm -rf output-distr
mkdir output-distr
hdfs dfs -get /user/$USER/output/part* output-distr

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

Expected Output:

module load

hadoop/2.6.0rm -

rf output

intermediate

hadoop --config $HOME jar multiply.jar edu.uta.cse6331.Multiply M-matrix-small.txt N-matrix-small.txt intermediate output

EXPERIMENT 6

AIM: To Develop a MapReduce to find the maximum electrical consumption in each year given electrical consumption for each month in each year.

Given below is the data regarding the electrical consumption of an organization. It Contains themonthly electrical consumption and the annual average for various years.

If the above data is given as input, we have to write applications to process it and produce resultssuch as finding the year of maximum usage, year of minimum usage, and so on. This is a walkover for the programmers with finite number of records. They will simply write the logic to produce the required output, and pass the data to the application written.

But, think of the data representing the electrical consumption of all the largescale industries of aparticular state, since its formation.

When we write applications to process such bulk data,

- They will take a lot of time to execute.
- There will be a heavy network traffic when we move data from source to network server andso on.

To solve these problems, we have the MapReduce framework

Input Data

The above data is saved as sample.txt and given as input. The input file looks as shown below.

1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	00	40	39	39	45

Source code:

```
import java.util.*;
import
java.io.IOException;
import
java.io.IOException;
import
org.apache.hadoop.fs.Path;
import
```

```
org.apache.hadoop.conf.*;
import
org.apache.hadoop.io.*;

import
org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
public class ProcessUnits
{
//Mapper class
public static class E_EMapper extends MapReduceBase implements
Mapper&lt;LongWritable ,/*Input key Type */ Text,                /*Input value
Type*/Text,    /*Output key Type*/ IntWritable&gt;            /*Output value
Type*/
{
//Map function
public void map(LongWritable key, Text value, OutputCollector&lt;Text, IntWritable&gt; output,
Reporter reporter) throws IOException
{
String line = value.toString(); String lasttoken =
null;StringTokenizer s = new
StringTokenizer(line,"\t"); String year =
s.nextToken(); while(s.hasMoreTokens())
{
lasttoken=s.nextToken();
}
int avgprice = Integer.parseInt(lasttoken);
output.collect(new Text(year), new
IntWritable(avgprice));
}
}
//Reducer class
public static class E_EReduce extends MapReduceBase
implementsReducer&lt; Text, IntWritable, Text, IntWritable &gt;
{
//Reduce function
public  void reduce( Text key, Iterator &lt;IntWritable&gt; values, OutputCollector&lt;Text,
IntWritable&gt; output, Reporter reporter) throws
IOException
{
int maxavg=30;
int
val=Integer.MIN_VAL
UE;while
(values.hasNext())
{
if((val=values.next().get())&gt;maxavg)
{
output.collect(key, new IntWritable(val));
}
}
}
```

```
}
//Main function
public static void main(String args[])throws Exception
{
JobConf conf = new JobConf(ProcessUnits.class);
conf.setJobName("max_eletricityunits");
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(E_EMapper.class);
conf.setCombinerClass(E_EReduce.class);
conf.setReducerClass(E_EReduce.class);
conf.setInputFormat(TextInputFormat.class);
conf.setOutputFormat(TextOutputFormat.class);
FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
JobClient.runJob(conf);
}
```

Expected
OUTPUT:Input:

Kolkata,56 Jaipur,45 Delhi,43 Mumbai,34
Goa,45 Kolkata,35 Jaipur,34 Delhi,32
Output: Kolkata 56
Jaipur 45
Delhi 43
Mumbai 34

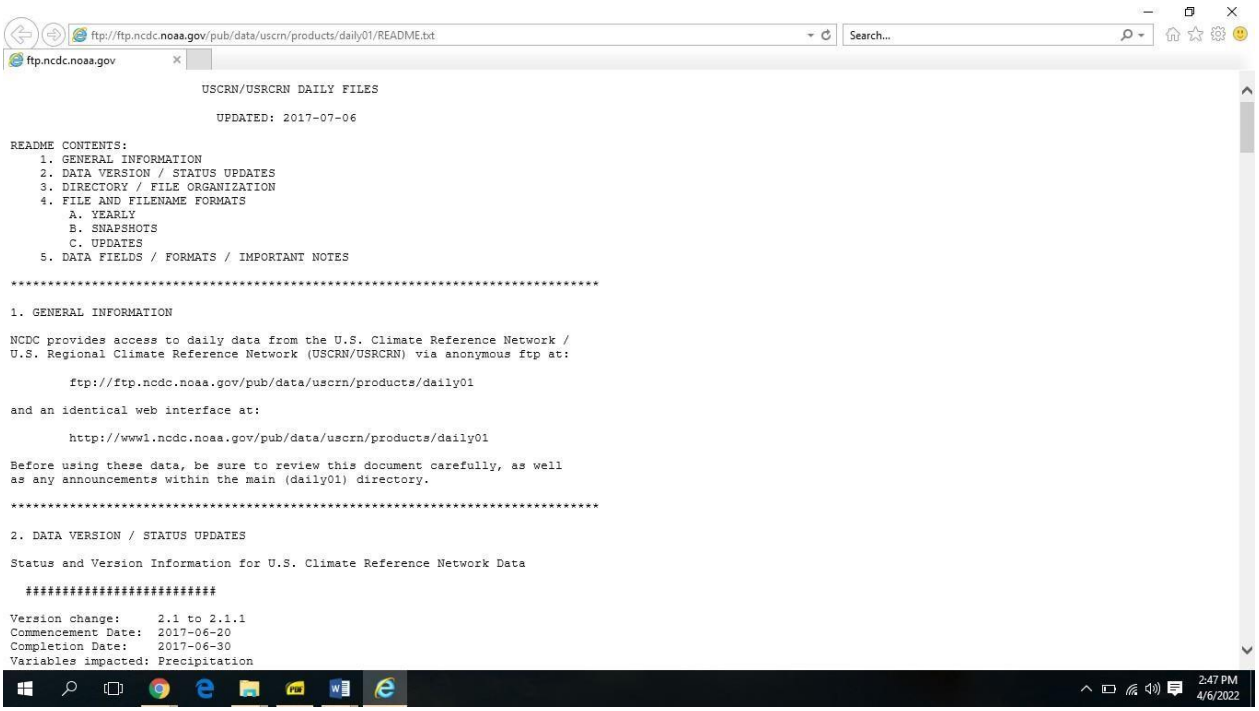
EXPERIMENT 7

AIM: To Develop a MapReduce to analyze weather data set and print whether the day is shinny or cool day.

NOAA's National Climatic Data Center (***NCDC***) is responsible for preserving, monitoring, assessing, and providing public access to weather data.

NCDC provides access to daily data from the U.S. Climate Reference Network / U.S. Regional Climate Reference Network (USCRN/USRCRN) via anonymous ftp at:

Dataset ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01



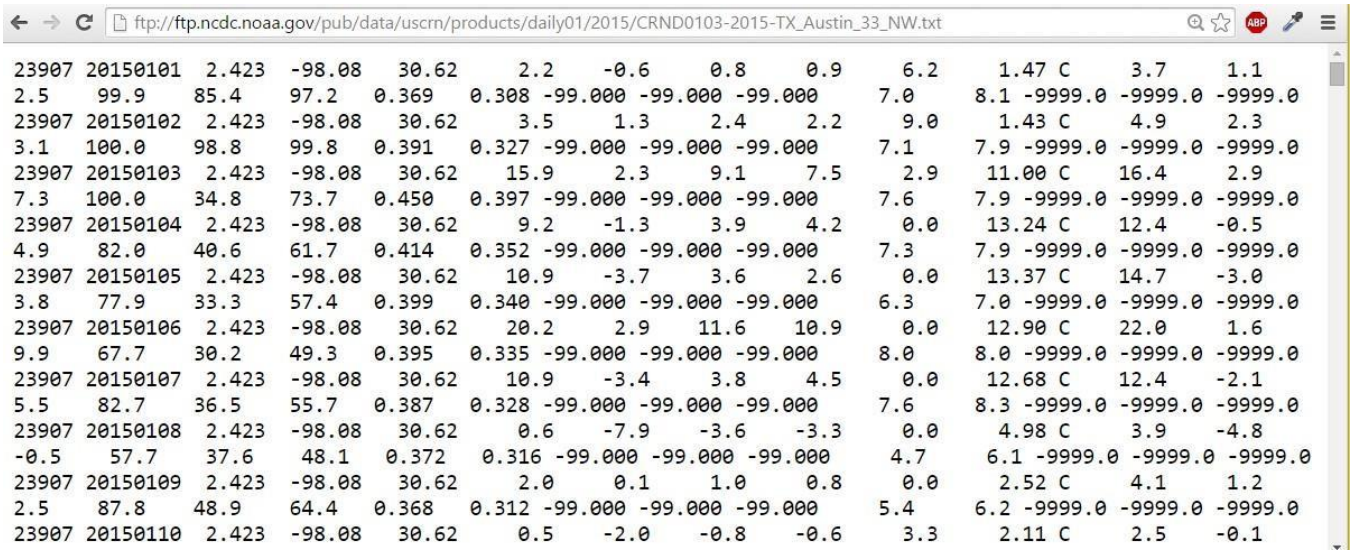
After going through wordcount mapreduce guide, you now have the basic idea of how a mapreduce program works. So, let us see a complex mapreduce program on weather dataset. Here I am using one of the dataset of year 2015 of Austin, Texas . We will do analytics on the dataset and classify whether it was a hot day or a cold day depending on the temperature recorded by NCDC.

NCDC gives us all the weather data we need for this

mapreduce project. The dataset which we will be using looks

like below snapshot.

ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/2015/CRND0103-2015- TX_Austin_33_NW.txt



Step 1

Download the complete project using below link.

https://drive.google.com/file/d/0B2SFMPvhXPQ5bUdoVFZsQjE2ZDA/view?usp=sharing

```
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 {
public static class MaxTemperatureMapper extends
Mapper<LongWritable, Text, Text, Text> {
 /**
 * @method map
* This method takes the input as 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 > 35  and
temp_min < 10 are passed to the reducer.
*/ @Override
public void map(LongWritable arg0, Text Value, Context 2 context) throws IOException,
InterruptedException {
 //Converting the record (single line) to String and storing it in a String variable line
String line = Value.toString();
//Checking if the line is not empty
 if (!(line.length() == 0)) {
 //date
String date = line.substring(6, 14);
//maximum temperature
float temp_Max = Float
       parseFloat(line.substring(39, 45).trim());
//minimum temperature
float temp_Min = Float
parseFloat(line.substring(47, 53).trim());
//if maximum temperature is greater than 35 , its a hot day
```

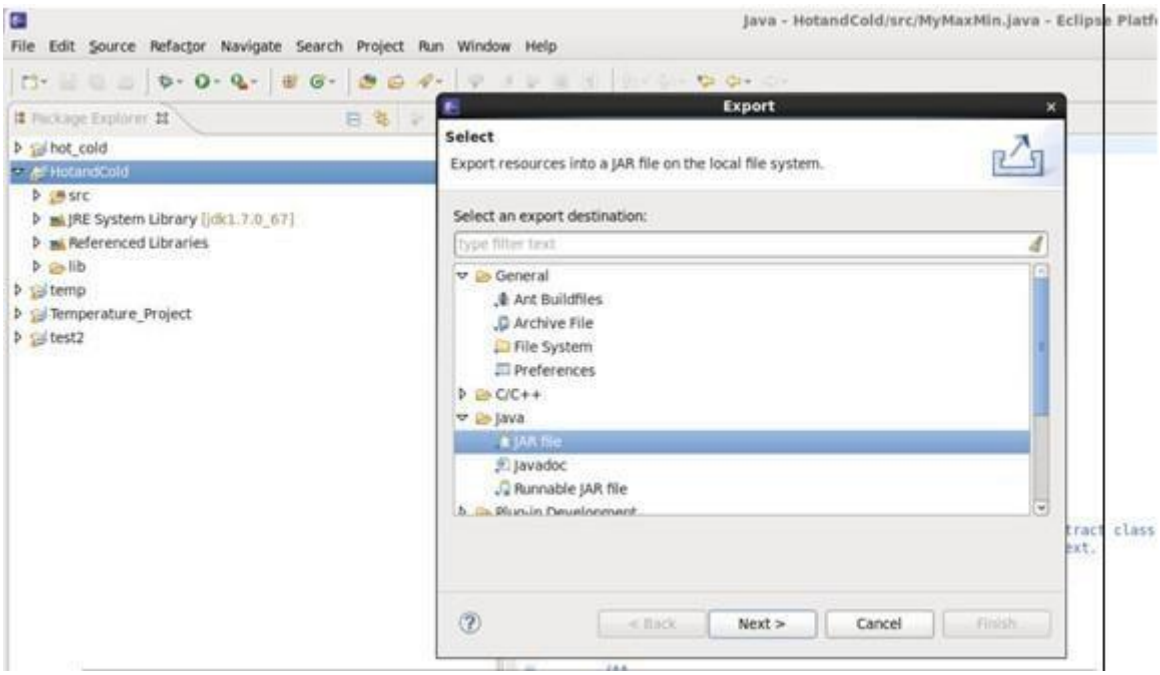
```
if (temp_Max > 35.0) {
// Hot day
context.write(new Text("Hot Day " + date),
new Text(String.valueOf(temp_Max)));
}
//if minimum temperature is less than 10, it's a cold day

if (temp_Min < 10) {

        // Cold day

context.write(new Text("Cold Day " + date),

new  Text(String.valueOf(temp_Min)));
}
}
}
}
//Reducer
*MaxTemperatureReducer class is static and extends Reducer abstract

having four hadoop generics type Text, Text, Text, Text.

*/

public static class MaxTemperatureReducer extends Reducer<Text, Text, Text,
Text> {

public void reduce (Text Key, Iterator<Text> Values, Context context) throws
IOException, Interrupted Exception {
String temperature = Values.next().toString();
context.write(Key, new Text(temperature));
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
```

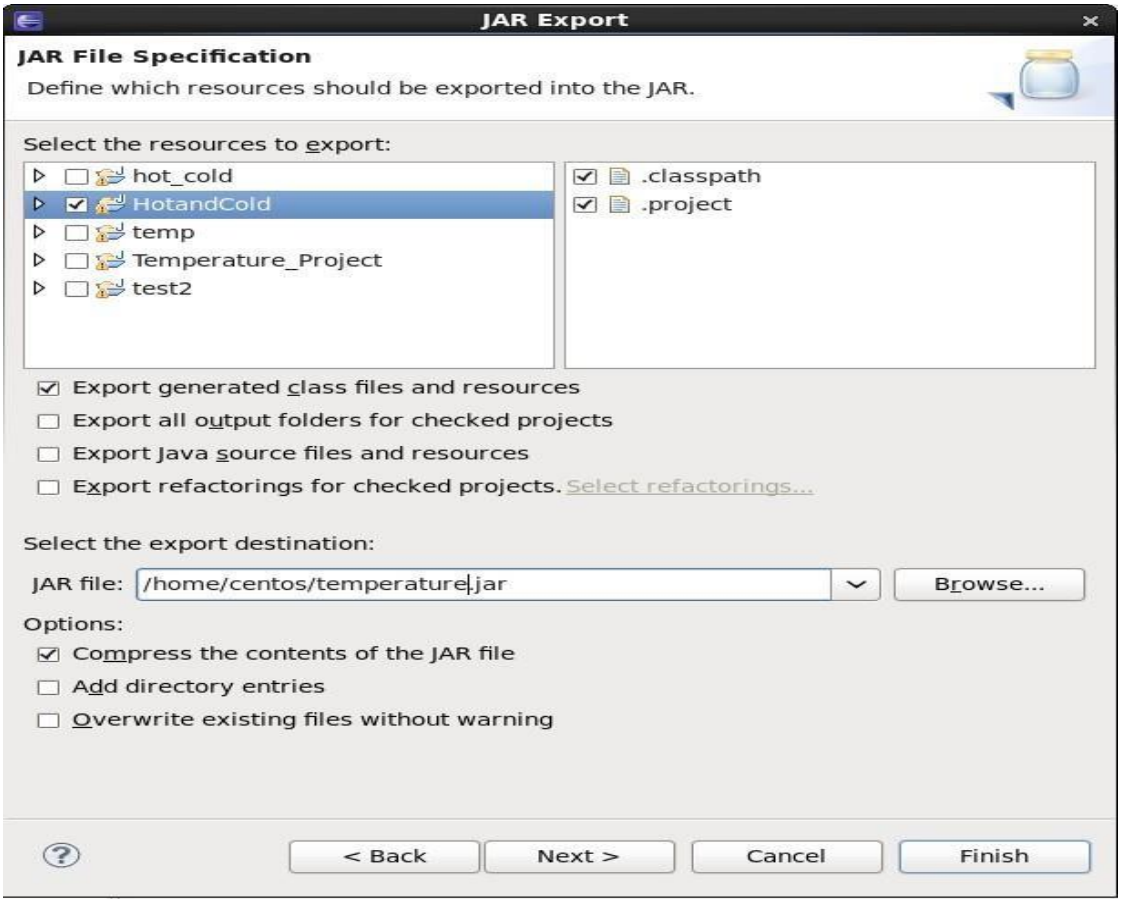
```
Job job = new Job(conf, "weather example");

job.setJarByClass(MyMaxMin.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);
job.setMapperClass(MaxTemperatureMapper.class);

job.setReducerClass(MaxTemperatureReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

Path OutputPath = new Path(args[1]);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

OutputPath.getFileSystem(conf).delete(OutputPath);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}
```

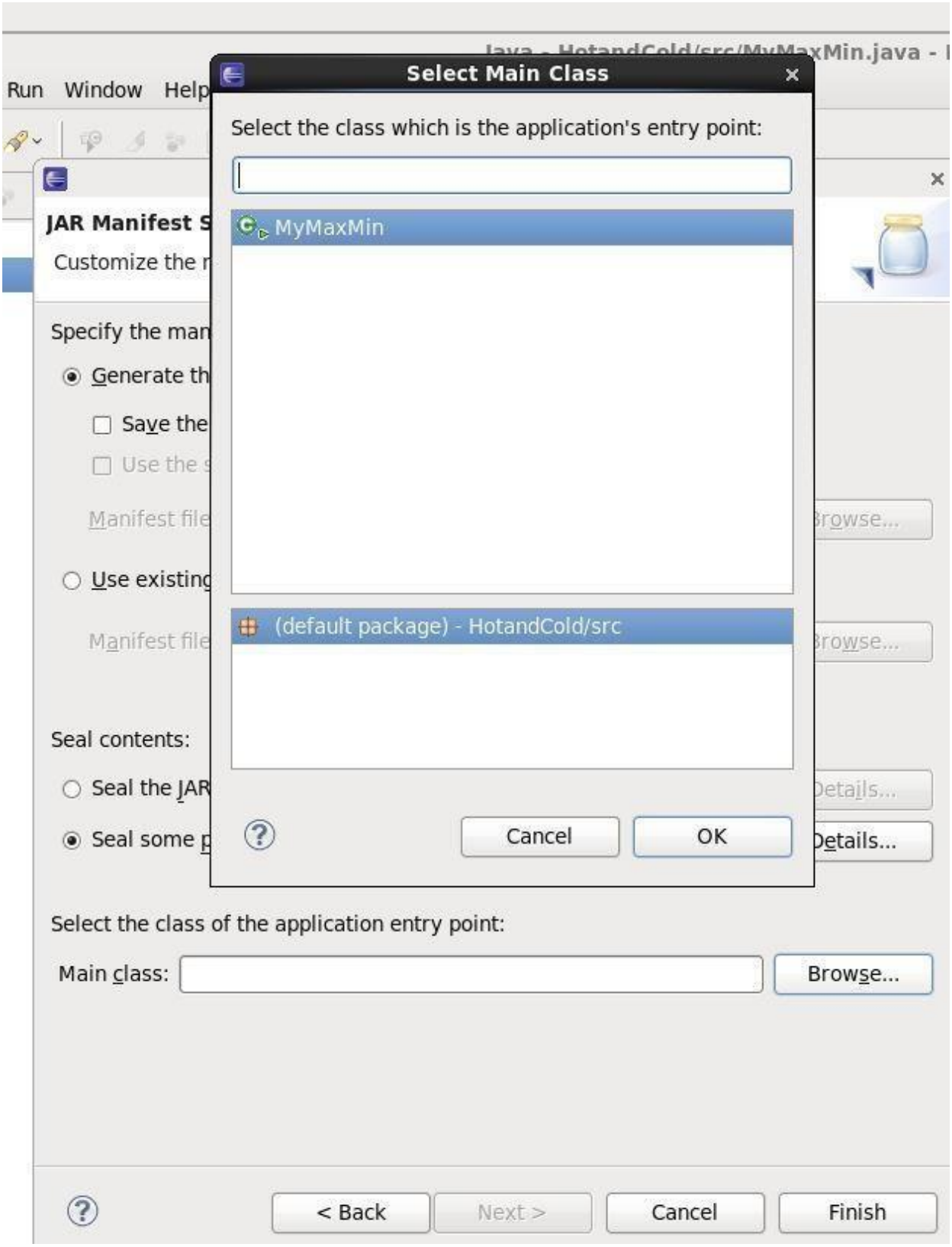
Import the project in eclipse IDE in the same way it was told in earlier guide and change the jar paths with the jar files present in the lib directory of this project.

When the project is not having any error, we will export it as a jar file, same as we did in wordcount mapreduce guide. Right Click on the Project file and click on Export. Select jar file.

Give the path where you want to save the file.

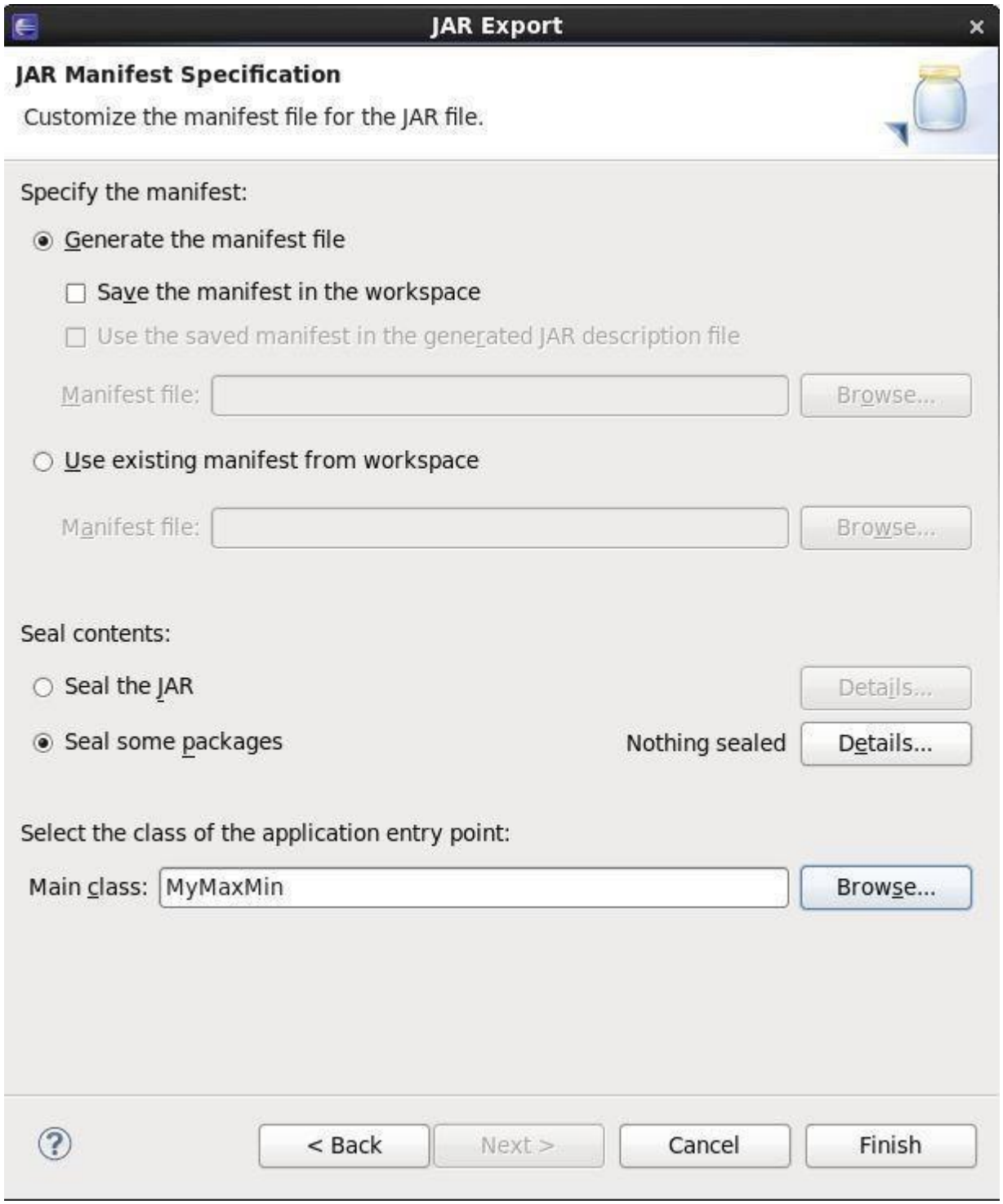

Java - HotandCold/src/MyMaxMin.java - Eclipse Platf
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
hot_cold
HotandCold
src
JRE System Library [jdk1.7.0_67]
Referenced Libraries
lib
temp
Temperature_Project
test2
Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
type filter text
General
Ant Buildfiles
Archive File
File System
Preferences
C/C++
Java
JAR file
Javadoc
Runnable JAR file
< Back
Next >
Cancel
Finish


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
hot_cold
HotandCold
temp
Temperature_Project
test2
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: /home/centos/temperature.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Cancel
Finish



NAME - PREETI ENROLLMENT NO. - 04215611922 SECTION – T11

Click on Finish to export.



You can download the jar file directly using below link

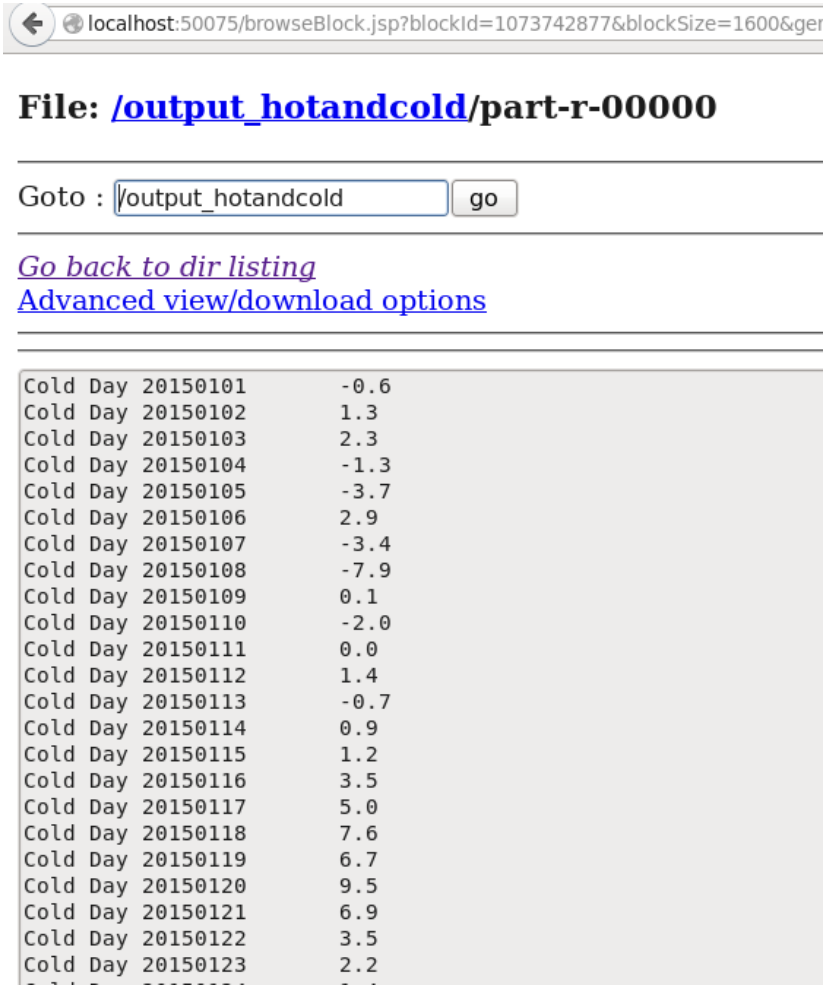
temperature.jar

https://drive.google.com/file/d/0B2SFMPvhXPQ5RUlZZDZSR3FYVDA/view?usp=sharing

Download Dataset used by me using below

link weather_data.txt

https://drive.google.com/file/d/0B2SFMPvhXPQ5aFVILXAxbFh6ejA/view?usp=sharing



EXPERIMENT 8

AIM: To Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.

For this analysis the Microsoft R Open distribution was used. The reason for this was its multithreaded performance as described here. Most of the packages that were used come from the tidyverse - a collection of packages that share common philosophies of tidy data. The tidytext and wordcloud packages were used for some text processing. Finally, the doMC package was used to embrace the multithreading in some of the custom functions which will be described later.
doMC package is not available on Windows. Use doParallel package instead.

Driver1.java

```
package KPI_1;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class Driver1
{
        public static void main(String[] args) throws Exception {
                Path firstPath = new Path(args[0]);
                Path sencondPath = new Path(args[1]);

                Path outputPath_1 = new Path(args[2]);
                Path outputPath_2 = new Path(args[3]);

                Configuration conf = new Configuration();
                 Job job = Job.getInstance(conf, "Most Viewed Movies");
```

```
                //set Driver class
                job.setJarByClass(Driver1.class);

            //output format for mapper
                job.setMapOutputKeyClass(LongWritable.class);
                job.setMapOutputValueClass(Text.class);

            //output format for reducer
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(LongWritable.class);

            //use MultipleOutputs and specify different Record class and Input formats
                MultipleInputs.addInputPath(job, firstPath, TextInputFormat.class,
movieDataMapper.class);
                MultipleInputs.addInputPath(job, sencondPath, TextInputFormat.class,
ratingDataMapper.class);
                //set Reducer class
                job.setReducerClass(dataReducer.class);
                FileOutputFormat.setOutputPath(job, outputPath_1);
                job.waitForCompletion(true)
                  Job job1 = Job.getInstance(conf, "Most Viewed Movies2");
                        job1.setJarByClass(Driver1.class);
                    //set Driver class
                    //set Mapper class
                    job1.setMapperClass(topTenMapper.class);
                    //set reducer class
                    job1.setReducerClass(topTenReducer.class);
                //output format for mapper
                    job1.setMapOutputKeyClass(Text.class);
                    job1.setMapOutputValueClass(LongWritable.class);
                    job1.setOutputKeyClass(LongWritable.class);
                    job1.setOutputValueClass(Text.class);

                    FileInputFormat.addInputPath(job1, outputPath_1);

                    FileOutputFormat.setOutputPath(job1, outputPath_2);
                    job1.waitForCompletion(true);
        }
```

```
}
```

dataReducer.java

```
import java.io.*;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class dataReducer extends Reducer<LongWritable,Text,Text,LongWritable>{

// here we are getting input from  ***movieDataMapper*** and
***userDataMapper***
        @Override
        public void reduce(LongWritable key, Iterable<Text>values,Context
context)throws IOException,InterruptedException
        {
                //key(movie_id)         values
                //234                 [ 1, ToyStory,1,1,1,1..... ]
                long count = 0;
                String movie_name = null;
                for(Text val:values)
                {
                        String token = val.toString();

                        if(token.equals("1")) //means data from userDataMapper
                        {
                                count++;

                        }
                        else



                        {
                                movie_name = token;  //means data from
movieDataMapper;
                        }
                }

                 context.write(new Text(movie_name), new LongWritable(count));
```

```
        }
}
```

movieDataMapper.java

```
import java.io.*;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class movieDataMapper extends Mapper <Object,Text,LongWritable,Text>{

        //data format => MovieID::Title::Genres

        @Override
        public void map(Object key,Text value,Context context)throws
IOException,InterruptedException
        {
                String []tokens = value.toString().split("::");
                long movie_id = Long.parseLong(tokens[0]);
                String name = tokens[1];
                context.write(new LongWritable(movie_id), new Text(name));
                        //movie_id              name
        }
}
```

ratingDataMapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class ratingDataMapper extends Mapper<Object,Text,LongWritable,Text> {
	//data format => UserID::MovieID::Rating::Timestamp
		@Override
		public void map(Object key,Text value,Context context)throws
IOException,InterruptedException
		{

			String []tokens = value.toString().split("::");
			long movie_id = Long.parseLong(tokens[1]);
			String count = "1";
			context.write(new LongWritable(movie_id), new Text(count));
		 // movie_id		count
	}
}
```

topTenMapper.java

```
import java.io.*;
import java.util.*;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;

public class topTenMapper extends Mapper<Object,Text,Text,LongWritable> {

	private TreeMap<Long,String> tmap;
	String movie_name=null;
	long count=0;
	@Override
	public void setup(Context context)throws IOException, InterruptedException
	{
	 tmap = new TreeMap<Long,String>();
	}
```

```
	@Override
	public void map(Object key,Text value,Context context)throws
IOException,InterruptedException
	{
		//data format => movie_name   count     (tab delimited) from dataReducer
		String []tokens = value.toString().split("\t");
		count = Long.parseLong(tokens[1]);
		movie_name = tokens[0].trim();
		tmap.put(count, movie_name);
		if(tmap.size() >10)		//if size crosses 10 we will remove the
topmost key-value pair.
		{
			tmap.remove(tmap.firstKey());
		}
	}
	@Override
	public void cleanup(Context context) throws IOException,InterruptedException
	{
		for(Map.Entry<Long,String> entry : tmap.entrySet()) {

			Long key = entry.getKey();		//count
			 String value = entry.getValue();	//movie_name

			 context.write(new Text(value),new LongWritable(key));
		}
	}
}
```

topTenReducer.java

```
import java.io.*;
import java.util.Map;
import java.util.TreeMap;

import org.apache.hadoop.io.LongWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class topTenReducer extends Reducer <Text,LongWritable,LongWritable,Text> {

        private TreeMap<Long,String> tmap2;
        String movie_name=null;
        long  count=0;
        @Override
        public void setup(Context context)throws IOException, InterruptedException
        {
         tmap2 = new TreeMap<Long,String>();
        }
        @Override
        public void reduce(Text key, Iterable<LongWritable> values,Context
context)throws IOException,InterruptedException
        {
                //data format => movie_name     count
                for(LongWritable val:values)
                {
                count = val.get();
                }
                movie_name = key.toString().trim();
                tmap2.put(count,movie_name);
                if(tmap2.size()>10)
                {
                        tmap2.remove(tmap2.firstKey());             }
        }
        @Override
        public void cleanup(Context context) throws IOException,InterruptedException
        {
                for(Map.Entry<Long,String> entry : tmap2.entrySet())
                {
                   Long key = entry.getKey();             //count
```

```
                    String value = entry.getValue();      //movie_name

                  context.write(new LongWritable(key),new Text(value));
                }

        }

}
```

EXPERIMENT 9

AIM: To Develop a MapReduce program to analyze Uber data set to find the days on which each basement has more trips using the following dataset.

The Uber dataset consists of four columns they are

dispatching_base_number	date	active_vehicles	trips

Problem Statement 1: In this problem statement, we will find the days on which each basement has more trips.

Source Code

Mapper Class:

```
public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
java.text.SimpleDateFormat format = new
java.text.SimpleDateFormat("MM/dd/yyyy");String[] days
={"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};
private Text basement = new
Text();Date date = null;
private int trips;
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException
{String line = value.toString();
String[] splits = line.split(",");
basement.set(splits[0]);

try {
date = format.parse(splits[1]);
} catch (ParseException e) {
// TODO Auto-generated catch
blocke.printStackTrace();
}
trips = new Integer(splits[3]);
String keys = basement.toString()+ "
"+days[date.getDay()];context.write(new Text(keys), new
IntWritable(trips));
```

```
}
}
```

Reducer Class:

```
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable>
{private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable>
values,Context context
) throws IOException, InterruptedException
{int sum = 0
for (IntWritable val : values)
{sum += val.get();
}
result.set(sum);
context.write(key,
result);
}
}
```

Whole Source Code:

```
import java.io.IOException;
import
java.text.ParseException;
import java.util.Date;
import
org.apache.hadoop.conf.Configuration;
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.output.FileOutputFormat;

public class Uber1 {
public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
java.text.SimpleDateFormat format = new
java.text.SimpleDateFormat("MM/dd/yyyy");String[] days
```

```
={"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};
private Text basement = new
Text();Date date = null;
private int trips;
public void map(Object key, Text value, Context context

) throws IOException, InterruptedException
{String line = value.toString();
String[] splits = line.split(",");
basement.set(splits[0]);
try {
date = format.parse(splits[1]);
} catch (ParseException e) {
// TODO Auto-generated catch
blocke.printStackTrace();
}
trips = new Integer(splits[3]);
String keys = basement.toString()+ "
"+days[date.getDay()];context.write(new Text(keys), new
IntWritable(trips));
}
}
public static class IntSumReducer
extends
Reducer<Text,IntWritable,Text,IntWritable>private
IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable>
valuesContext context
) throws IOException, InterruptedException
{int sum = 0;
for (IntWritable val : values)
{sum += val.get();
}
result.set(sum);
context.write(key,
result);
}
}
public static void main(String[] args) throws Exception
{Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "Uber1");

job.setJarByClass(Uber1.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
```

```
job.setOutputValueClass(IntWritable.class);


FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new
Path(args[1]));System.exit(job.waitForCompletion(true)
? 0 : 1);
}
}
```

Running the Program:

First, we need to build a jar file for the above program and we need to run it as a normal Hadoopprogram by passing the input dataset and the output file path as shown below.
hadoop jar uber1.jar /uber /user/output1
In the output file directory, a part of the file is created and contains the below

Expected Output:

```
B02512 Sat 15026
B02512 Sun 10487
B02512 Thu 15809
B02512 Tue 12041
B02512 Wed 12691
B02598 Fri 93126
B02598 Mon 60882
B02598 Sat 94588
B02598 Sun 66477
B02598 Thu 90333
B02598 Tue 63429
B02598 Wed 71956
B02617 Fri 125067
B02617 Mon 80591
B02617 Sat 127902
B02617 Sun 91722
B02617 Thu 118254
B02617 Tue 86602
B02617 Wed 94887
B02682 Fri 114662
B02682 Mon 74939
B02682 Sat 120283
B02682 Sun 82825
B02682 Thu 106643
B02682 Tue 76905
B02682 Wed 86252

B02764 Fri 326968
B02764 Mon 214116
```

B02764 Sat 356789
B02764 Sun 249896
B02764 Thu 304200
B02764 Tue 221343
B02764 Wed 241137
B02765 Fri 34934
B02765 Mon 21974
B02765 Sat 36737

EXPERIMENT 10

AIM: Develop a MapReduce program to analyze Titanic ship data and to find the average age of the people (both male and female) who died in the tragedy. How many persons are survived in each class.

The titanic data will be..

Column 1 :PassengerI d	Column 2 : Survived (survived=0 &died=1)
Column 3 :Pclass	Column 4 : Name
Column 5 : Sex	Column 6 : Age
Column 7 :SibSp	Column 8 :Parch
Column 9 : Ticket	Column 10 : Fare
Column 11 :Cabin	Column 12 : Embarked

Description:

There have been huge disasters in the history of Map reduce, but the magnitude of the Titanic'sdisaster ranks as high as the depth it sank too. So much so that subsequent disasters have alwaysbeen described as "titanic in proportion" – implying huge losses.

Anyone who has read about the Titanic, know that a perfect combination of natural events and human errors led to the sinking of the Titanic on its fateful maiden journey from Southampton toNew York on April 14, 1912.

There have been several questions put forward to understand the cause/s of the tragedy – foremost among them is: What made it sink and even more intriguing How can a 46,000 ton ship sink to the depth of 13,000 feet in a matter of 3 hours? This is a mind boggling question indeed!

There have been as many inquisitions as there have been questions raised and equally that many types of analysis methods applied to arrive at conclusions. But this blog is not about analyzing why or what made the Titanic sink – it is about analyzing the data that is present about the Titanic publicly. It actually uses Hadoop MapReduce to analyze and arrive at:

- The average age of the people (both male and female) who died in the tragedyusing Hadoop MapReduce.
- How many persons survived – traveling class wise.

This blog is about analyzing the data of Titanic. This total analysis is performed in HadoopMapReduce.

This Titanic data is publically available and the Titanic data set is described below under the heading Data Set Description.

Using that dataset we will perform some Analysis and will draw out some insights like finding the average age of male and females died in Titanic, Number of males and females died in each compartment.

DATA SET DESCRIPTION

Column 1 : PassengerI
Column 2 : Survived (survived=0 & died=1) Column 3 : Pclass
Column 4 : Name
Column 5 : Sex
Column 6 : Age
Column 7 : SibSp
Column 8 : Parch
Column 9 : Ticket
Column 10 : Fare
Column 11 : Cabin
Column 12 : Embarked

Mapper code:

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

{private Text gender = new Text();

private IntWritable age = new IntWritable();

public void map(LongWritable key, Text value, Context context )
throwsIOException, InterruptedException {

String line =

value.toString();String

str[]=line.split(",");

if(str.length>6){

gender.set(str[4]);

if((str[1].equals("0")) ){


if(str[5].matches("\\d+")){
```

```
int
i=Integer.parseInt(str[5]);
age.set(i);
}
}
}
context.write(gender, age)
}
}
```

Reducer Code:

```
public static class Reduce extends Reducer<Text,IntWritable, Text, IntWritable>
{public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
int sum = 0;
int l=0;
for (IntWritable val : values)
{l+=1;
sum += val.get();
}
sum=sum/l;
context.write(key, new IntWritable(sum));
}
}
```

Configuration Code:

```
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
```

https://github.com/kiran0541/Map-Reduce/blob/master/Average%20age%20of%20male%20and%20female%20people%20died%20in%20titanic

Way to to execute the Jar file to get the result of the first problem statement:

hadoop jar average.jar /TitanicData.txt /avg_out

Here 'hadoop' specifies we are running a Hadoop command and jar specifies

which type of application we are running and average.jar is the jar file which we have created which consists the above source code and the path of the Input file name in our case it is TitanicData.txt and the output file where to store the output here we have given it as avg out.

Way to view the output:

hadoop dfs –cat /avg_out/part-r-00000

Here 'hadoop' specifies that we are running a Hadoop commandand 'dfs' specifies that we are performing an operation related to HadoopDistributed File System and '- cat' is used to view the contents of a file and'avg_out/part-r-00000' is the file where the output is stored. Part file is created by default by the TextInputFormat class of Hadoop.

EXPERIMENT 11

AIM: To Develop a program to calculate the maximum recorded temperature by yearwise for the weather dataset in Pig Latin

Description:

The National Climatic Data Center (NCDC) is the world's largest active archive of weather data. I downloaded the NCDC data for year 1930 and loaded it in HDFS system. I implemented MapReduce program and Pig, Hove scripts to findd the Min, Max, avg temparature for diffrentstations.

Compiled the Java File: javac -classpath /home/student3/hadoop-common-2.6.1.jar:/home/student3/hadoop-mapreduce-client-core-2.6.1.jar:/home/student3/commons-cli- 2.0.jar -d . MaxTemperature.java MaxTemperatureMapper.java MaxTemperatureReducer.java

Created the JAR file: jar -cvf hadoop-project.jar *class

Executed the jar file: hadoop jar hadoop-project.jar MaxTemperature /home/student3/Project/ /home/student3/Project_output111

Copy the output file to local hdfs dfs -copyToLocal /home/student3/Project_output111/part-r- 00000

PIG Script

Pig -x local grunt> records = LOAD '/home/student3/Project/Project_Output/output111.txt' AS (year:chararray, temperature:int); grunt> DUMP records; grunt> grouped_records = GROUP records BY year; grunt> DUMP grouped_records; grunt> max_temp = FOREACH grouped_records GENERATE group,

Hive Script

Commands to create table in hive and to find average

temperatureDROP TABLE IF EXISTS w_hd9467;

CREATE TABLE w_hd9467(year STRING, temperature INT) ROW FORMAT DELIMITEDFIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH '/home/student3/Project/Project_Output/output1.txt'

OVERWRITE INTO TABLE w_hd9467;

SELECT count(*) from w_hd9467;

SELECT * from w_hd9467 limit 5;

Query to find average temperature SELECT year, AVG(temperature) FROM w_hd9467 GROUP BY year;

MaxTemperature.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.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception
    {if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output
      path>");System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new
    Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

MaxTemperatureMapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import
```

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

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

  private static final int MISSING = 9999;

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

    String line = value.toString();
    String year = line.substring(15,
    19);int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus
      signsairTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new
      IntWritable(airTemperature));
    }
  }
}
```

MaxTemperatureReducer.java

```
import java.io.IOException;
import
org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import

org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer

  extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable>
     values,Context context)
     throws IOException, InterruptedException {
```

```
    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
    context.write(key, value);
     // maxValue = Math.max(maxValue, value.get());
    }
    //context.write(key, new IntWritable(maxValue));
  }
 }
```

Expected Output:

```
1921  -222
1921  -144
1921  -122
1921  -139
1921  -122
1921  -89
1921  -72
1921  -61
1921  -56
1921  -44
1921  -61
1921  -72
1921  -67
1921  -78
1921  -78
1921  -133
1921  -189
1921  -250
1921  -200
1921  -150
1921  -156
```

1921 -144
1921 -133
1921 -139
1921 -161
1921 -233
1921 -139
1921 -94
1921 -89
1921 -122
1921 -100
1921 -100
1921 -106
1921 -117
1921 -144
1921 -128
1921 -139
1921 -106
1921 -100
1921 -94
1921 -83
1921 -83
1921 -106
1921 -150
1921 -200
1921 -178
1921 -72
1921 -156

EXPERIMENT 12

AIM: To Write queries to sort and aggregate the data in a table using HiveQL

Description:

Hive is an open-source data warehousing solution built on top of Hadoop. It supports an SQL-likequery language called HiveQL. These queries are compiled into MapReduce jobs that are executed on Hadoop. While Hive uses Hadoop for execution of queries, it reduces the effort that goes into writing and maintaining MapReduce jobs.

Hive supports database concepts like tables, columns, rows and partitions. Both primitive (integer, float, string) and complex data-types(map, list, struct) are supported. Moreover, these types can be composed to support structures of arbitrary complexity. The tables are serialized/deserialized usingdefault serializers/deserializer. Any new data format and type can be supported by implementing SerDe and ObjectInspector java interface.

HiveQL - ORDER BY and SORT BY Clause

By using HiveQL ORDER BY and SORT BY clause, we can apply sort on the column. It returns the result set either in ascending or descending order. Here, we are going to execute these clauseson the records of the below table:

emp

Id	Name	Salary	Department
1	Gaurav	30000	Developer
2	Aryan	20000	Manager
3	Vishal	40000	Manager
4	John	10000	Trainer
5	Henry	25000	Developer
6	William	9000	Developer
7	Lisa	25000	Manager
8	Ronit	20000	Trainer

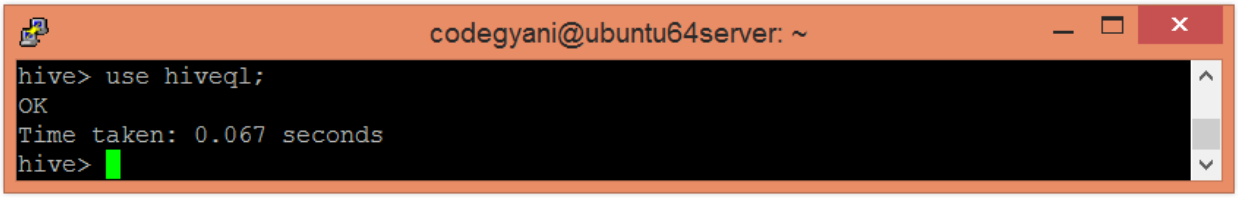
HiveQL - ORDER BY Clause

In HiveQL, ORDER BY clause performs a complete ordering of the query result set. Hence, the complete data is passed through a single reducer. This may take much time in the execution of large datasets. However, we can use LIMIT to minimize the sorting time.

Example:

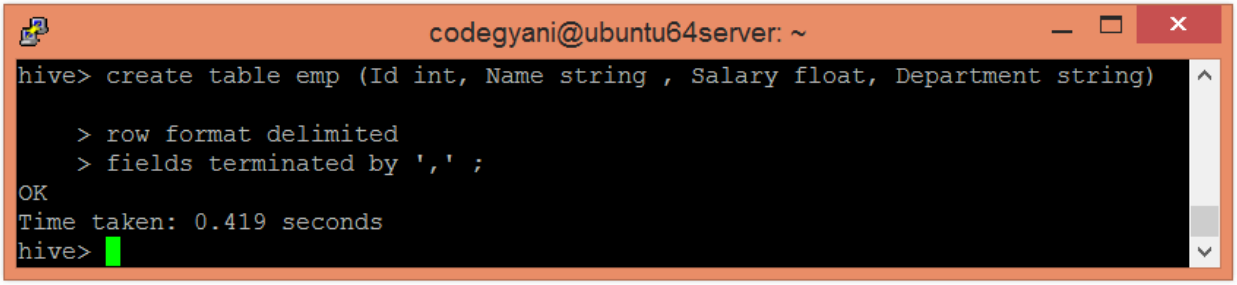
Select the database in which we want to create a table.

hive> use hiveql;



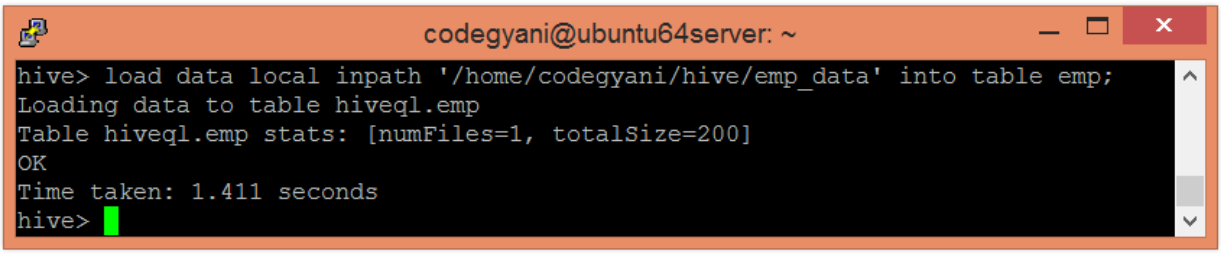
Now, create a table by using the following command:

hive> create table emp (Id int, Name string , Salary float, Department string)row format delimited
fields terminated by ',' ;



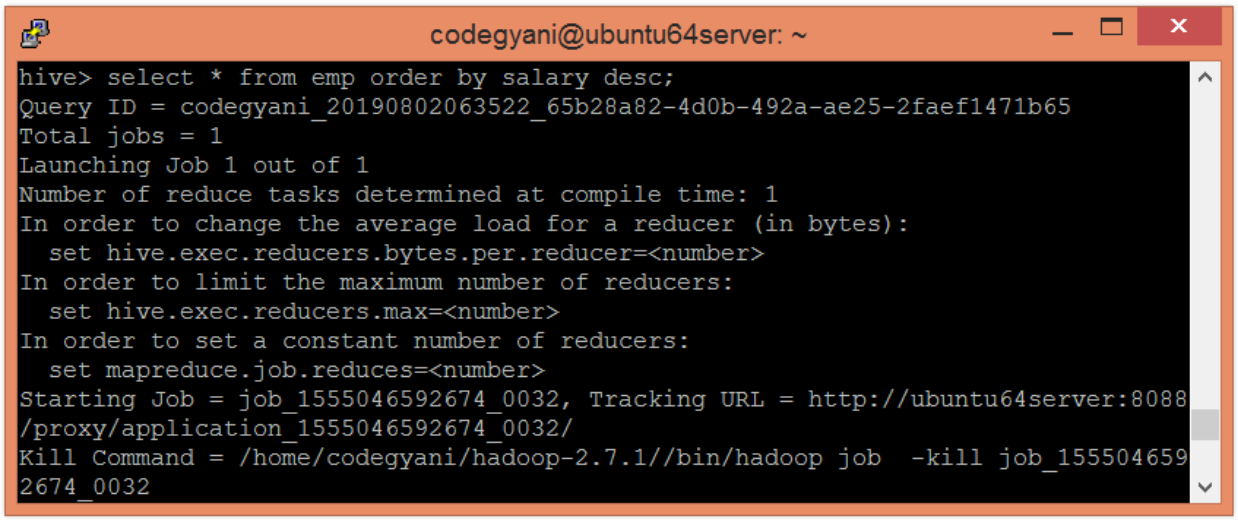
Load the data into the table

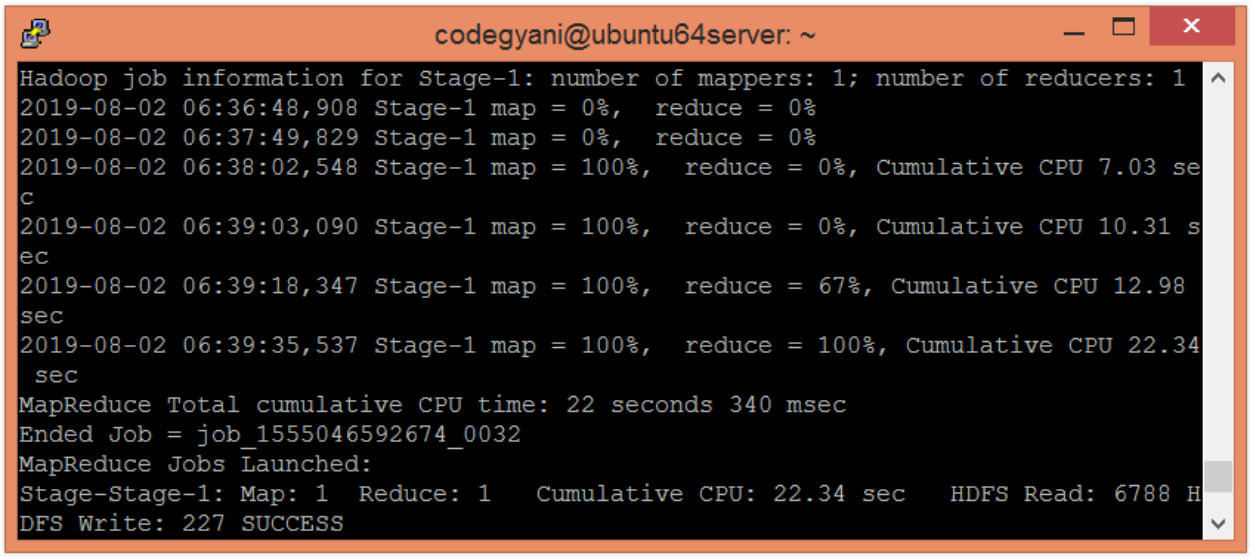
hive> load data local inpath '/home/codegyani/hive/emp_data' into table emp;

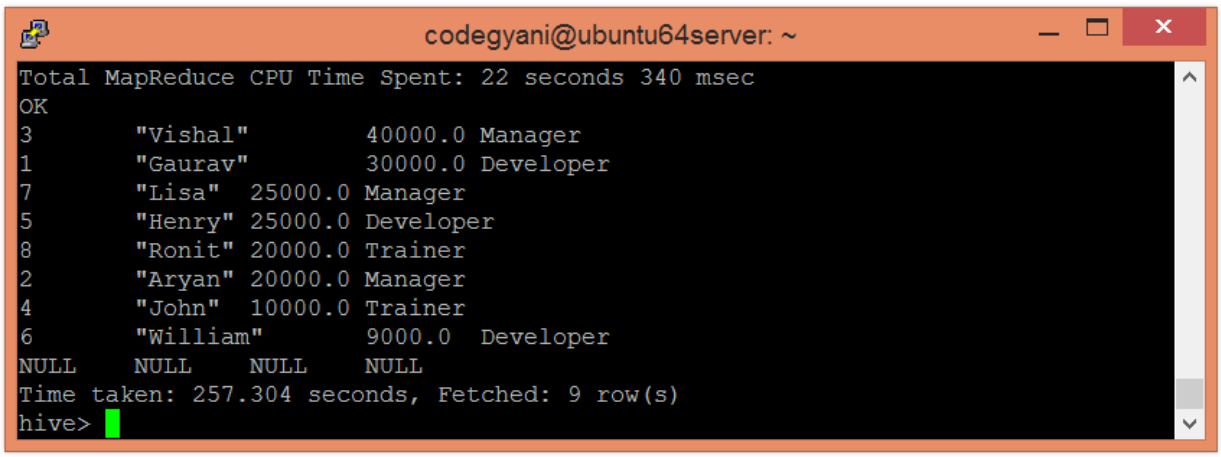


Now, fetch the data in the descending order by using the following

commandhive> select * from emp order by salary desc;







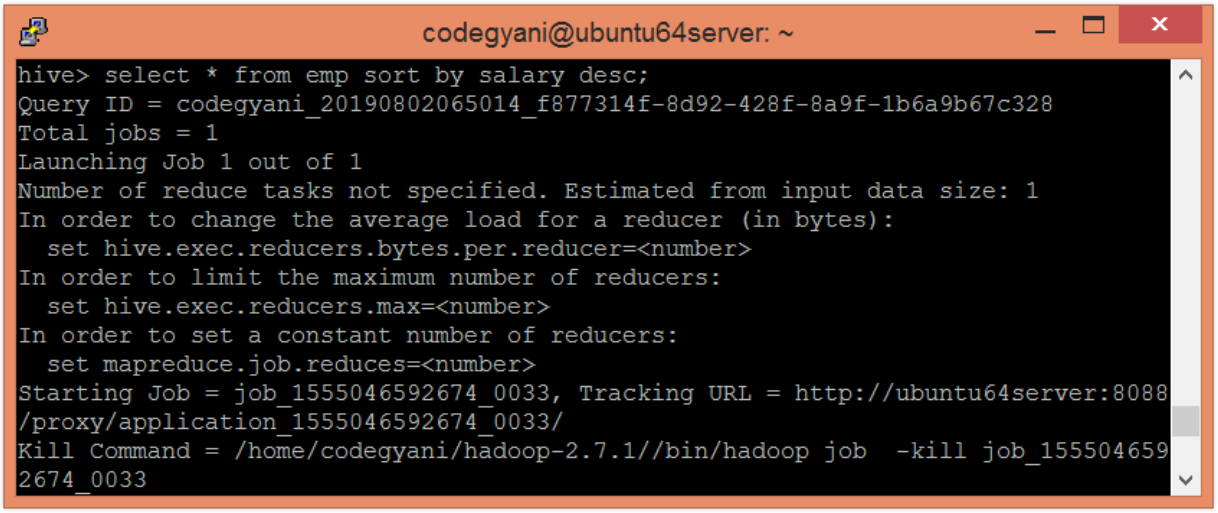
HiveQL - SORT BY Clause

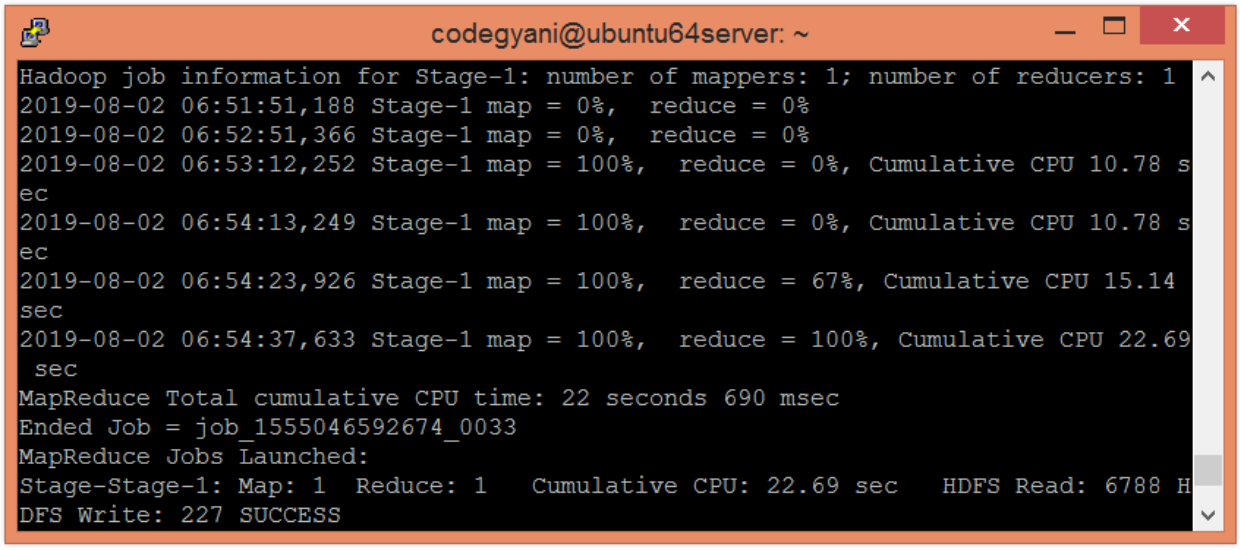
The HiveQL SORT BY clause is an alternative of ORDER BY clause. It orders the data within each reducer. Hence, it performs the local ordering, where each reducer's output is sorted separately.It may also give a partially ordered result.

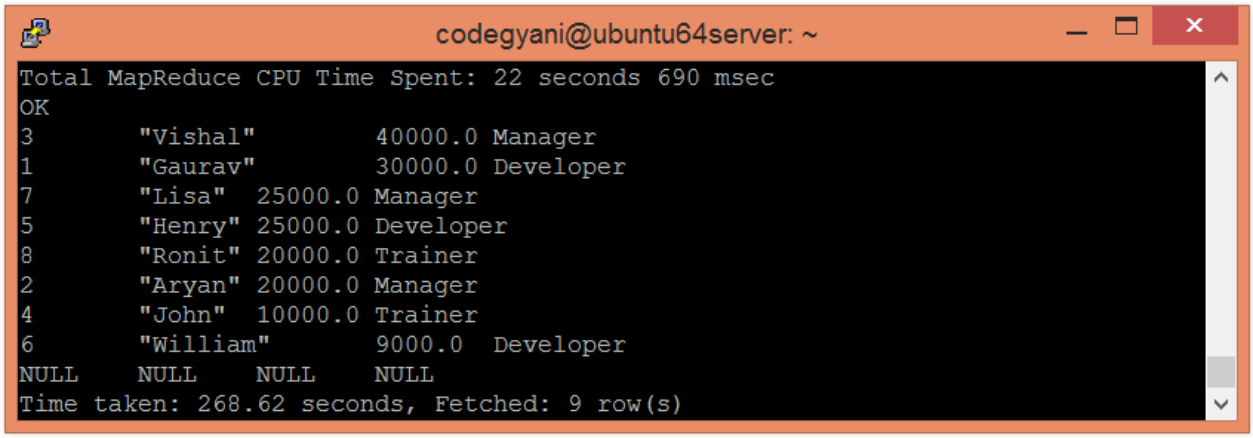
Example:

Let's fetch the data in the descending order by using the following command

hive> select * from emp sort by salary desc;







Cluster By:

Cluster By used as an alternative for both Distribute BY and Sort BY clauses in Hive-QL.

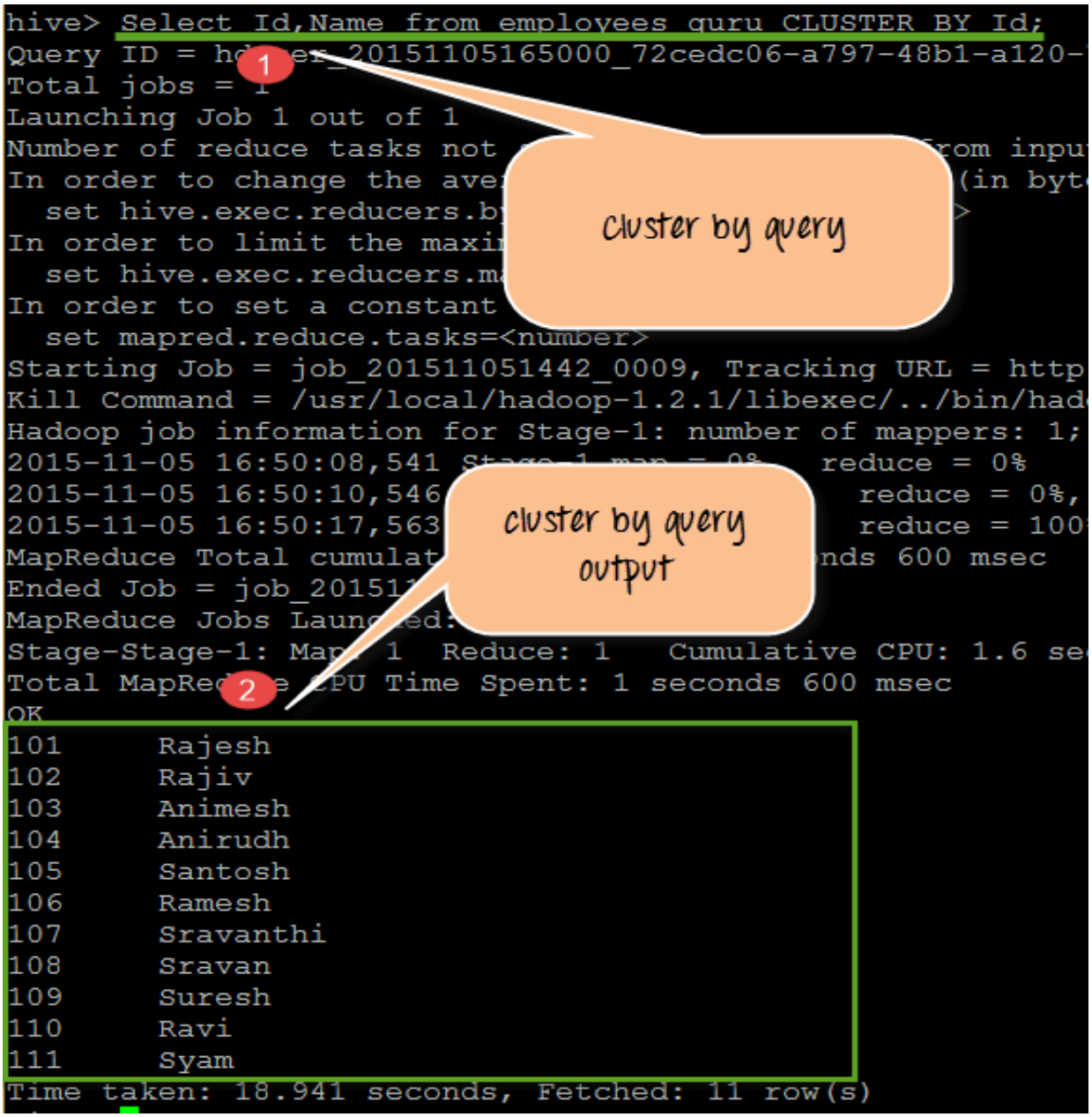
Cluster BY clause used on tables present in Hive. Hive uses the columns in Cluster by todistribute the rows among reducers. Cluster BY columns will go to the multiple reducers.

- It ensures sorting orders of values present in multiple reducers

For example, Cluster By clause mentioned on the Id column name of the table employees_guru table. The output when executing this query will give results to multiple reducers at the back end.But as front end it is an alternative clause for both Sort By and Distribute By.

Example:

SELECT Id, Name from employees_guru CLUSTER BY Id;


hive> Select Id,Name from employees guru CLUSTER BY Id;
Query ID = h
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not
In order to change the ave
set hive.exec.reducers.b
In order to limit the maxi
set hive.exec.reducers.m
In order to set a constant
set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0009, Tracking URL = http
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/had
Hadoop job information for Stage-1: number of mappers: 1;
2015-11-05 16:50:08,541
2015-11-05 16:50:10,546
2015-11-05 16:50:17,563
MapReduce Total cumulat
Ended Job = job_201511
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.6 se
Total MapRe CPU Time Spent: 1 seconds 600 msec
OK
101 Rajesh
102 Rajiv
103 Animesh
104 Anirudh
105 Santosh
106 Ramesh
107 Sravanthi
108 Sravan
109 Suresh
110 Ravi
111 Syam
Time taken: 18.941 seconds, Fetched: 11 row(s)
1
cluster by query
cluster by query output
2

SECTION B(VALUE ADDED EXPERIMENTS)

EXPERIMENT 1

OBJECTIVE:

a. Run the Pig Latin Scripts to find Word Count.
b. Run the Pig Latin Scripts to find a max temp for each and every year.

PROGRAM LOGIC:

Run the Pig Latin Scripts to find Word Count.

```
lines = LOAD '/user/hadoop/HDFS_File.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group,
COUNT(words);DUMP wordcount;
```

Run the Pig Latin Scripts to find a max temp for each and every year

```
-- max_temp.pig: Finds the maximum
temperature by yearrecords = LOAD
'input/ncdc/micro-tab/sample.txt'
AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND
(quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality
== 9);grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE
group,MAX(filtered_records.temperature);
DUMP max_temp;
```

OUTPUT:

```
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
(1949,78,1)
```

```
    }
}
```

Expected Output:

GDP per capita, 2013

Country	GDP in billions of US dollars	Population in millions	Per capita GDP in US dollars
Brazil	2,246.00	199.20	11,172.50
Canada	1,826.80	35.10	52,037.10
China	9,469.10	1,360.80	6,958.70
Egypt	271.40	83.70	3,242.90
Germany	3,636.00	80.80	44,999.50
India	1,876.80	1,243.30	1,509.50
Japan	4,898.50	127.3	38,467.80
Mexico	1,260.90	118.40	10,649.90
South Korea	1,304.47	50.20	25,975.10
United Kingdom	2,523.20	64.10	39,371.70
United States	16,768.10	316.30	53,001.00

EXPERIMENT 2

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

RESOURCES:

VMWare, XAMPP Server, Web Browser, 1GB RAM, Hard Disk 80 GB.

PROGRAM LOGIC:

SYNTAX for HIVE Database
OperationsDATABASE Creation
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
Drop Database Statement
DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS]
database_name
[RESTRICT|CASCADE];Creating
and Dropping Table in HIVE
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]
table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment] [ROW FORMAT row_format] [STORED
ASfile_format]
Loading Data into table
log_dataSyntax:
LOAD DATA LOCAL INPATH '<path>/u.data' OVERWRITE INTO TABLE
u_data;
Alter Table
in HIVE
Syntax
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec
...])ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name
new_type ALTER TABLE name REPLACE COLUMNS (col_spec[,
col_spec ...])Creating and Dropping View
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT
column_comment], ...)] [COMMENT table_comment] AS SELECT ...
Droppin
g View
Syntax:
DROP VIEW
view_name
Functions in HIVE
String Functions:- round(), ceil(), substr(), upper(), reg_exp() etc

Date and Time Functions:- year(), month(), day(),
to_date() etcAggregate Functions :- sum(), min(),
max(), count(), avg() etc INDEXES
CREATE INDEX index_name ON TABLE base_table_name (col_name,
...)AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES
(property_name=property_value, ...)][IN TABLE
index_table_name]
[PARTITIONED BY (col_name,
...)][
[ROW FORMAT ...] STORED AS ...
| STORED BY ...
]
[LOCATION
hdfs_path]
[TBLPROPERTIES
(...)]
Creating Index
CREATE INDEX index_ip ON TABLE log_data(ip_address) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH
DEFERREDREBUILD;
Altering and Inserting Index
ALTER INDEX index_ip_address ON log_data
REBUILD;Storing Index Data in Metastore
SET
hive.index.compact.file=/home/administrator/Desktop/big/metastore_db/tmp/index
_ipadd ress_result;
SET
hive.input.format=org.apache.hadoop.hive.ql.index.compact.HiveCompactInde
xInputFor mat;
Dropping Index
DROP INDEX INDEX_NAME on TABLE_NAME;