BIG DATA ANALYTICS LABORATORY

1. Install Apache Hadoop

Introduction

Apache Hadoop is an open-source software framework used to store, manage and process large datasets for various big data computing applications running under clustered systems. It is Java-based and uses Hadoop Distributed File System (HDFS) to store its data and process data using MapReduce. In this article, you will learn how ro install and configure Apache Hadoop on Ubuntu 20.04.

Prerequisites

- Deploy a fully updated Vultr Ubuntu 20.04 Server.
- Create a non-root user with sudo access.

1.1 Install Java

Install the latest version of Java.

```
$ sudo apt install default-jdk default-jre -y
```

Verify the installed version of Java.

```
$ java -version
```

1.2 Create Hadoop User and Configure Password-less SSH

Add a new user hadoop.

```
$ sudo adduser hadoop
```

Add the hadoop user to the sudo group.

```
$ sudo usermod -aG sudo hadoop
```

Switch to the created user.

```
$ sudo su - hadoop
```

Install the OpenSSH server and client.

```
$ apt install openssh-server openssh-client -y
```

When you get a prompt, respond with:

```
keep the local version currently installed
```

Switch to the created user.

```
$ sudo su - hadoop
```

Generate public and private key pairs.

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

Add the generated public key from id_rsa.pub to authorized_keys.

```
$ sudo cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Change the permissions of the authorized_keys file.

```
$ sudo chmod 640 ~/.ssh/authorized_keys
```

Verify if the password-less SSH is functional.

```
$ ssh localhost
```

1.3 Install Apache Hadoop

Log in with hadoop user.

```
$ sudo su - hadoop
```

download hadoop from url :

```
$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
```

Extract the downloaded file.

```
$ tar -xvzf hadoop-3.3.1.tar.gz
```

Move the extracted directory to the /usr/local/ directory.

```
$ sudo mv hadoop-3.3.1 /usr/local/hadoop
```

Create directory to store system logs.

```
$ sudo mkdir /usr/local/hadoop/logs
```

Change the ownership of the hadoop directory.

```
$ sudo chown -R hadoop:hadoop /usr/local/hadoop
```

1.4 Configure Hadoop

Edit file ~/.bashrc to configure the Hadoop environment variables.

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

Add the following lines to the file. Save and close the file.

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

```
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

Activate the environment variables.

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

1.5 Configure Java Environment Variables

Hadoop has a lot of components that enable it to perform its core functions. To configure these components such as YARN, HDFS, MapReduce, and Hadoop-related project settings, you need to define Java environment variables in hadoop-env.sh configuration file.

Find the Java path.

```
$ which javac
```

Find the OpenJDK directory.

```
$ readlink -f /usr/bin/javac
```

Edit the hadoop-env.sh file.

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

Add the following lines to the file. Then, close and save the file.

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_CLASSPATH+=" $HADOOP_HOME/lib/*.jar"
```

Browse to the hadoop lib directory.

```
$ cd /usr/local/hadoop/lib
```

Download the Javax activation file.

```
$ sudo wget https://jcenter.bintray.com/javax/activation/javax.activation-api/
1.2.0/javax.activation-api-1.2.0.jar
```

Verify the Hadoop version.

```
$ hadoop version
```

Edit the core-site.xml configuration file to specify the URL for your NameNode.

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

Add the following lines. Save and close the file.

```
<configuration>
   <property>
      <name>fs.default.name</name>
      <value>hdfs://0.0.0.0:9000</value>
      <description>The default file system URI</description>
   </property>
</configuration>
```

Create a directory for storing node metadata and change the ownership to hadoop.

```
$ sudo mkdir -p /home/hadoop/hdfs/{namenode,datanode}

$ sudo chown -R hadoop:hadoop /home/hadoop/hdfs
```

Edit hdfs-site.xml configuration file to define the location for storing node metadata, fs-image file.

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

Add the following lines. Close and save the file.

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

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

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

Edit mapred-site.xml configuration file to define MapReduce values.

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

Add the following lines. Save and close the file.

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

Edit the yarn-site.xml configuration file and define YARN-related settings.

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

Add the following lines. Save and close the file.

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

Log in with hadoop user.

```
$ sudo su - hadoop
```

Validate the Hadoop configuration and format the HDFS NameNode.

```
$ hdfs namenode -format
```

1.6 Start the Apache Hadoop Cluster

Start the NameNode and DataNode.

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

Start the YARN resource and node managers.

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

Verify all the running components.

```
$ jps
```

1.7 Access Apache Hadoop Web Interface

You can access the Hadoop NameNode on your browser via http://server-IP:9870. For example:

```
http://127.0.0.1:9870
```

Conclusion

You have successfully installed Apache Hadoop on your server. You can now access the dashboard and configure your preferences.

2. Develop a MapReduce program to calculate the frequency of a given word in agiven file.

WordCount.java

```
import java.io.IOException;

import java.util.StringTokenizer;

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 WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

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

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.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);
  }
}
```

creating data :

-> create a file name “file” using nano command : nano file

-> enter the below text :

```
Hello World Bye World Hello Hadoop Goodbye Hadoop
```

commands for execution :

```
$ hadoop com.sun.tools.javac.Main WordCount.java

$ jar cf wc.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/wordcount/input/

$ hadoop fs -put -f file /user/hadoop/wordcount/input/

$ hadoop fs -ls /user/hadoop/wordcount/input/

$ hadoop fs -cat /user/hadoop/wordcount/input/file

$ hadoop jar wc.jar WordCount /user/hadoop/wordcount/input
/user/hadoop/wordcount/output

$ hadoop fs -cat /user/hadoop/wordcount/output/part-r-00000
```

output :

```
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

3. Develop a MapReduce program to find the maximum temperature in each year.

MaxMonTem.java

```
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 MaxMonTem {
        //Mapper class
        public static class MM_TMapper extends MapReduceBase implements
        Mapper<LongWritable, /*Input key*/
        Text,
        Text,
        IntWritable> {
                //Map function
                public void map (LongWritable key, Text value,
                OutputCollector<Text, IntWritable>output,
                Reporter reporter) throws IOException {

                        String line = value.toString();
                        StringTokenizer st = new StringTokenizer(line,")(");
                        while(st.hasMoreTokens()){
                        String token = st.nextToken();
                        if(token.length() > 2){
                        StringTokenizer sst = new StringTokenizer(token,",");
                        String year = sst.nextToken().substring(0,4);
                        int tem = Integer.parseInt(sst.nextToken());
                        output.collect(new Text(year), new IntWritable(tem));
                        }
                }
           }
        }
 //Reducer class
public static class MM_TReduce extends MapReduceBase implements
        Reducer<Text, IntWritable, Text, IntWritable> {
                //Reduce function
                public void reduce(Text key, Iterator <IntWritable> values,
                        OutputCollector<Text,IntWritable> output, Reporter
reporter) throws IOException {
                        int max = values.next().get();
                        while(values.hasNext()){
                                int tem = values.next().get();
                                if(max < tem){
                                        max = tem;
                                }
                        }
                        output.collect(key, new IntWritable(max));
                }
        }
        public static void main(String[] args) throws Exception{
                JobConf conf = new JobConf(MaxMonTem.class);
```

```
                conf.setJobName("maximumMonthly_temperature");
                conf.setOutputKeyClass(Text.class);
                conf.setOutputValueClass(IntWritable.class);
                conf.setMapperClass(MM_TMapper.class);
                conf.setCombinerClass(MM_TReduce.class);
                conf.setReducerClass(MM_TReduce.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);
        }
}
```

data set : data_tem.txt

(200707,100), (200706,90)

(200508,90), (200607,100)

(200708,80), (200606,80)

commands for execution :

```
$ hadoop com.sun.tools.javac.Main MaxMonTem.java

$ jar cf mmt.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/MaxMonTem/input/

$ hadoop fs -put -f data_tem.txt /user/hadoop/MaxMonTem/input/

$ hadoop fs -ls /user/hadoop/MaxMonTem/input/

$ hadoop fs -cat /user/hadoop/MaxMonTem/input/data_tem.txt

$ hadoop jar mmt.jar MaxMonTem /user/hadoop/MaxMonTem/input
/user/hadoop/MaxMonTem/output

$ hadoop fs -cat /user/hadoop/MaxMonTem/output/part-r-00000
```

output :

```
2005    90

2006    100

2007    100
```

4. Develop a MapReduce program to implement Matrix Multiplication.

Map.java

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

import java.io.IOException;

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

MatrixMultiply.java

```
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MatrixMultiply {

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MatrixMultiply <in_dir> <out_dir>");
            System.exit(2);
        }
        Configuration conf = new Configuration();
        // M is an m-by-n matrix; N is an n-by-p matrix.
        conf.set("m", "1000");
        conf.set("n", "100");
        conf.set("p", "1000");
        @SuppressWarnings("deprecation")
                Job job = new Job(conf, "MatrixMultiply");
        job.setJarByClass(MatrixMultiply.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

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

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

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

Reduce.java

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

import java.io.IOException;
import java.util.HashMap;

public class Reduce
  extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text> {
        @Override
        public void reduce(Text key, Iterable<Text> values, Context context)
                        throws IOException, InterruptedException {
                String[] value;
                HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
                HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();
                for (Text val : values) {
                        value = val.toString().split(",");
                        if (value[0].equals("M")) {
                                hashA.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
                        } else {
                                hashB.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
                        }
                }
```

```
                int n = Integer.parseInt(context.getConfiguration().get("n"));
                float result = 0.0f;
                float m_ij;
                float n_jk;
                for (int j = 0; j < n; j++) {
                        m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
                        n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
                        result += m_ij * n_jk;
                }
                if (result != 0.0f) {
                        context.write(null,
                                        new Text(key.toString() + "," +
Float.toString(result)));
                }
        }
}
```

data set : M

```
M,1,2,10
M,3,2,9
M,6,3,9
```

data set : N

```
N,2,2,9
N,6,7,8
N,8,8,10
```

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf mm.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/mm/input/

$ hadoop fs -put -f M /user/hadoop/mm/input/

$ hadoop fs -put -f N /user/hadoop/mm/input/

$ hadoop fs -ls /user/hadoop/mm/input/

$ hadoop fs -cat /user/hadoop/mm/input/M

$ hadoop fs -cat /user/hadoop/mm/input/N

$ hadoop jar mm.jar  /user/hadoop/mm/input /user/hadoop/mm/output

$ hadoop fs -cat /user/hadoop/mm/output/part-r-00000
```

output :

```
999,970,493.0

999,971,586.0

999,972,763.0

999,973,717.0

999,974,236.0

999,975,532.0
```

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

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 {
```

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

public static final int MISSING = 9999;

@Override

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

String line = Value.toString();

if (!(line.length() == 0)) {

String date = line.substring(6, 14);

float temp_Max = Float.parseFloat(line.substring(39, 45).trim());

float temp_Min = Float.parseFloat(line.substring(47, 53).trim());

if (temp_Max > 30.0) {

context.write(new Text("The Day is Hot Day :" + date), new
Text(String.valueOf(temp_Max)));

}

if (temp_Min < 15) {

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

}

}

}

}

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

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

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);

}

}
```

download dataset :

https://www1.ncdc.noaa.gov/pub/data/uscrn/products/daily01/2021/CRND0103-2021-NV_Mercury_3_SSW.txt

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf whether.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/whether/input/

$ hadoop fs -put -f RND0103-2021-NV_Mercury_3_SSW.txt
/user/hadoop/whether/input/

$ hadoop fs -ls /user/hadoop/whether/input/
```

```
$ hadoop fs -cat /user/hadoop/whether/input/RND0103-2021-NV_Mercury_3_SSW.txt

$ hadoop jar whether.jar  /user/hadoop/whether/input /user/hadoop/whether/output

$ hadoop fs -cat /user/hadoop/whether/output/part-r-00000
```

output :

```
The Day is Cold Day :20210101 0.0

The Day is Cold Day :20210102 -3.2

The Day is Cold Day :20210103 -0.3

The Day is Cold Day :20210104 0.2

The Day is Cold Day :20210105 -2.4

The Day is Cold Day :20210106 1.4
```

6. Develop a MapReduce program to find the number of products sold in each country by considering sales data containing fields like

Tranction_Date	Product	Price	Payment_Type	Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude

SalesMapper.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.mapred.*;


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

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

```
        public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {


                String valueString = value.toString();

                String[] SingleCountryData = valueString.split(",");

                output.collect(new Text(SingleCountryData[7]), one);

        }

}
```

SalesCountryDriver.java

```
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapred.*;


public class SalesCountryDriver {

      public static void main(String[] args) {

            JobClient my_client = new JobClient();

            JobConf job_conf = new JobConf(SalesCountryDriver.class);

            job_conf.setJobName("SalePerCountry");

            job_conf.setOutputKeyClass(Text.class);

            job_conf.setOutputValueClass(IntWritable.class);

            job_conf.setMapperClass(SalesMapper.class);

            job_conf.setReducerClass(SalesCountryReducer.class);

            job_conf.setInputFormat(TextInputFormat.class);

            job_conf.setOutputFormat(TextOutputFormat.class);

            FileInputFormat.setInputPaths(job_conf, new Path(args[0]));

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

            my_client.setConf(job_conf);
```

```
            try {
                  JobClient.runJob(job_conf);
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

SalesCountryReducer.java

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

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {
      public void reduce(Text t_key, Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException
{
            Text key = t_key;
            int frequencyForCountry = 0;
            while (values.hasNext()) {
                  IntWritable value = (IntWritable) values.next();
                  frequencyForCountry += value.get();
            }
            output.collect(key, new IntWritable(frequencyForCountry));
      }
}
```

download dataset :

https://drive.google.com/uc?export=download&id=1tP8AJGSgDXwI12r2Ap07GyamMj1o0iDD

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf sales.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/sales/input/

$ hadoop fs -put -f SalesJan2009.csv /user/hadoop/sales/input/

$ hadoop fs -ls /user/hadoop/sales/input/

$ hadoop fs -cat /user/hadoop/whether/input/SalesJan2009.csv

$ hadoop jar sales.jar SalesCountryDriver /user/hadoop/sales/input
/user/hadoop/sales/output

$ hadoop fs -cat /user/hadoop/sales/output/part-00000
```

output :

```
Argentina 1

Australia 38

Austria   7

Bahrain   1

Belgium   8

Bermuda   1

Brazil    5

Bulgaria  1
```

7.Develop a MapReduce program to find the min and max marks of student's.
ClassScore.java

```
import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;
import java.io.*;
import java.util.*;
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.Partitioner;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.util.GenericOptionsParser;

public class ClassScore {
    private static String SPACE = "\t";
    public static class Map extends

    Mapper<LongWritable, Text, Text, Text> {

        //Implement map function
        private Text word1=new Text("score");

        private Text word2=new Text("distribution");
        public void map(LongWritable key, Text value, Context context)

        throws IOException, InterruptedException {

             String line = value.toString();

             StringTokenizer tokenizerArticle = new
StringTokenizer(line);
            while (tokenizerArticle.hasMoreElements()) {
                String  strName = tokenizerArticle.nextToken();//score
part
                String  strScore = tokenizerArticle.nextToken();
                String  namescore=strName+SPACE+strScore;
                context.write(word1,new Text(namescore));
                context.write(word2,new Text(namescore));
            }

        }

    }


    public static class GenderPartitioner extends Partitioner<Text, Text>
{
```

```
        public int getPartition(Text key, Text value, int numReduceTasks)
{
            String[] namescore = value.toString().split(SPACE);

            int score = Integer.parseInt(namescore[1]);
            String str = key.toString();
            //Assign partition 0 by default
           //if (numReduceTasks == 0)
             //  return 0;
            if("distribution".equals(str))
            {
            if ((score>= 90)&&(score <=100)) {
                return 1 %  numReduceTasks ;
            }else if((score >=80)&&(score< 90)) {
                return 2 %  numReduceTasks;
            }else if((score >=70)&&(score< 80)){
                return 3 % numReduceTasks;
            }else if((score >= 60)&&(score<70)){
                return 4 % numReduceTasks;
            }else{
                return 5 % numReduceTasks;
            }
                }
            else
            {
                return 0;
            }
        }
    }

    public static class Reduce extends

    Reducer<Text, Text, Text, IntWritable> {
        public void reduce(Text key, Iterable<Text> values,

        Context context) throws IOException, InterruptedException {
           String  ss=key.toString();
            if("score".equals(ss))
            {
            int sum = 0;
            int count = 0;
            int min    =    150   ;
            int max    =    0   ;
            //int i=0;
            int score  =    0   ;
            String name1 =   " ";
            String name2 =   " ";
            List<String> cache =new ArrayList<String>();

            for (Text val : values) {
                cache.add(val.toString());
                String[] valTokens = val.toString().split(SPACE);
                score = Integer.parseInt(valTokens[1]);
                if (score > max) {
                    //name1 = valTokens[0];
                     max  = score;
                }
```

```
                if (score < min)
                {
                    min   =score;
                }
                sum+=score;
                count++;
            }
            int average = (int) sum/count;//calculate average score
            if(sum%count>=(count/2))
                average+=1;
            context.write(new Text("The average is"), new
IntWritable(average));

            context.write(new Text("The min score is"), new
IntWritable(min));

            for (String val : cache) {
                String[] valTokens = val.split(SPACE);
                score = Integer.parseInt(valTokens[1]);
                if(score==min)
                {
                     name2 = valTokens[0];
                    context.write(new Text(name2),new IntWritable(min));
                }
            }
            context.write(new Text("The max score is"), new
IntWritable(max));
            for (String val : cache) {
                String[] valTokens = val.split(SPACE);
                score = Integer.parseInt(valTokens[1]);
                if(score==max)
                {
                      name1 = valTokens[0];
                     context.write(new Text(name1),new
IntWritable(max));
                }
            }

            }
            else
            {
                 String nname  =" "  ;
                 int score     =0   ;
                for (Text val : values) {
                    String[] valTokens = val.toString().split(SPACE);
                    nname = valTokens[0];
                    score = Integer.parseInt(valTokens[1]);
                    context.write(new Text(nname), new
IntWritable(score));
                    }
            }
        }



    }
```

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

        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args)
                .getRemainingArgs();

        if (otherArgs.length != 2) { //Determine whether the path
parameter is 2

            System.err.println("Usage: Data Deduplication <in> <out>");

            System.exit(2);

        }

        //set maprduce job name
        Job job = new Job(conf, "ClassScore");

        job.setJarByClass(ClassScore.class);

        //Set the Map, Combine and Reduce processing classes

        job.setMapperClass(Map.class);



        job.setReducerClass(Reduce.class);

        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(IntWritable.class);

        job.setPartitionerClass(GenderPartitioner.class);

        job.setNumReduceTasks(6);

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

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

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

    }

}
```

data set : data.txt

```
ALLUGUNTI 50
RAKESH 35
SRAVANI 87
MANISHA 65
NIKHITHA 94
```

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf sg.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/sg/input/

$ hadoop fs -put -f data.txt /user/hadoop/sg/input/

$ hadoop fs -ls /user/hadoop/sg/input/

$ hadoop jar sg.jar /user/hadoop/sg/input /user/hadoop/sg/output

$ hadoop fs -cat /user/hadoop/sg/output/part-r-00000
```

output :

```
The average is        66
The min score is      35
RAKESH                35
The max score is      94
NIKHITHA              94
```

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

Elect_Driver.java

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


public class Elect_Driver {

	public static void main(String[] args) throws Exception {
		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "JobName");
		job.setJarByClass(Elect_Driver.class);
		job.setMapperClass(Elect_Mapper.class);
		job.setReducerClass(Elect_Reducer.class);
		// TODO: specify output types

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

		// TODO: specify input and output DIRECTORIES (not files)
		FileInputFormat.setInputPaths(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));

		if (!job.waitForCompletion(true))
			return;
	}

}
```

Elect_Mapper.java

```
import java.io.IOException;
import java.util.*;
import org.apache.commons.lang3.StringUtils;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.*;

public class Elect_Mapper extends Mapper<LongWritable, Text, IntWritable,
Text> {
```

```
    private IntWritable key=new IntWritable();


    int minmon=0;
    protected void setup(org.apache.hadoop.mapreduce.Mapper.Context
context)
            throws IOException, InterruptedException {

        super.setup(context);
        BufferedReader br=new BufferedReader( new InputStreamReader(new
FileInputStream("/home/hadoop/565/ec/Big-Data-Analytics/Mini
Project/elect.txt")));
        String s1;
        int i=0,c=0;
        int year=0;
        int jan=0;
        int feb=0;
        int mar=0,apr=0,may=0,jun=0,jul=0,aug=0,sep=0,oct=0,nov=0,dec=0;
        HashMap<String,String> hm=new HashMap();
        while((s1=br.readLine())!=null)
        {
           //System.out.println(s1);
        String s2[]=s1.split("   ");
      /* jan+=Integer.parseInt(s2[1]);
       feb+=Integer.parseInt(s2[2]);
       mar+=Integer.parseInt(s2[3]);
       apr+=Integer.parseInt(s2[4]);
       may+=Integer.parseInt(s2[5]);
       jun+=Integer.parseInt(s2[6]);
       jul+=Integer.parseInt(s2[7]);
       aug+=Integer.parseInt(s2[8]);
       sep+=Integer.parseInt(s2[9]);
       oct+=Integer.parseInt(s2[10]);
       nov+=Integer.parseInt(s2[11]);
       dec+=Integer.parseInt(s2[12]);*/

       //System.out.println(s2[1]);

//hm.put(s2[0],s2[1]+s2[2]+s2[3]+s2[4]+s2[5]+s2[6]+s2[7]+s2[8]+s2[9]+s2[1
0]+s2[11]+s2[12]);
       c++;
        }
        jan/=c;
        feb/=c;
        mar/=c;
        apr/=c;
        may/=c;
        jun/=c;
        jul/=c;
        aug/=c;
        sep/=c;
        oct/=c;
        nov/=c;
        dec/=c;

    }
    public void map(LongWritable ikey, Text ivalue, Context context)
```

```
            throws IOException, InterruptedException {
        String line=ivalue.toString();
        String[] tokens=StringUtils.split(line,' ');
        key.set(Integer.parseInt(tokens[0]));
        Text t = new Text(Integer.parseInt(tokens[1])+"
"+Integer.parseInt(tokens[2])
                +" "+Integer.parseInt(tokens[3])+"
"+Integer.parseInt(tokens[4])
                +" "+Integer.parseInt(tokens[5])+"
"+Integer.parseInt(tokens[6])
                +" "+Integer.parseInt(tokens[7])+"
"+Integer.parseInt(tokens[8])
                +" "+Integer.parseInt(tokens[9])+"
"+Integer.parseInt(tokens[10])
                +" "+Integer.parseInt(tokens[11])+"
"+Integer.parseInt(tokens[12]));
        //System.out.println(t);
        context.write(key,t);
    }

}
```

Elect_Reducer.java

```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;


public class Elect_Reducer extends
Reducer<IntWritable,Text,String,String> {

      static int maxelect=1000;
      static int max=0,min=500000;
       static int maxyear,minyear,mi=0,ma=0;
      static float avg=0;
      static int jan=0;
     static int feb=0;
     static int
mar=0,apr=0,may=0,jun=0,jul=0,aug=0,sep=0,oct=0,nov=0,dec=0;
     int a=0;
    public void reduce(IntWritable _key, Iterable<Text> values, Context
context)
            throws IOException, InterruptedException {
        // process values

        Iterator<Text> iterator=values.iterator();
        Text b=new Text(iterator.next());
        String br=""+b;
        String s[]=br.split(" ");


        //System.out.println(s[1]);
```

```
        int c=0;
        //while(c++!=1000)
       // {
           jan+=Integer.parseInt(s[0]);
           feb+=Integer.parseInt(s[1]);
           mar+=Integer.parseInt(s[2]);
           apr+=Integer.parseInt(s[3]);
           may+=Integer.parseInt(s[4]);
           jun+=Integer.parseInt(s[5]);
           jul+=Integer.parseInt(s[6]);
           aug+=Integer.parseInt(s[7]);
           sep+=Integer.parseInt(s[8]);
           oct+=Integer.parseInt(s[9]);
           nov+=Integer.parseInt(s[10]);
           dec+=Integer.parseInt(s[11]);
           a=Integer.parseInt(s[0])+Integer.parseInt(s[1])
+Integer.parseInt(s[2])
                      +Integer.parseInt(s[3])+Integer.parseInt(s[4])
+Integer.parseInt(s[5])
                      +Integer.parseInt(s[6])+Integer.parseInt(s[7])
+Integer.parseInt(s[8])
                      +Integer.parseInt(s[9])+Integer.parseInt(s[10])
+Integer.parseInt(s[11]);

       // }
        //System.out.println(jan+" "+feb+" "+mar+" "+apr+" "+may+"
"+jun+" "+jul+" "+aug+" "+sep+" "+oct+" "+nov+" "+dec);

        avg+=a;

        if(a>max)
           {max=a;
           maxyear=_key.get();
           //System.out.println(maxyear);
           }
        if(a<min)
     {min=a;
     minyear=_key.get();
     //System.out.println(maxyear);
     }
        maxelect--;



        if(maxelect==0)
        {

            IntWritable maxx=new IntWritable(max);
             IntWritable minn=new IntWritable(min);

           int arr[]={jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec};
           //System.out.println(arr[0]);
           String min=findmin(arr);
           String max=findmax(arr);
           String mini[]=min.split(" ");
           String maxi[]=max.split(" ");
        //System.out.println("t1"+maxyear);
```

```
        avg=avg/1000;
        //context.write(new IntWritable(),new IntWritable(avg));
        context.write("Year : "+new IntWritable(maxyear)+" has the
maximum annual consumption of "+maxx
        +"\nYear : "+new IntWritable(minyear)+" has the minimum annual
consumption of "+minn,

                 "\nThe month of "+mini[1]+" has the minimum total
consumption of "+mini[0]+" (monthwise)"
                 +"\nThe month of "+maxi[1]+" has the maximum total
consumption of "+maxi[0]+" (monthwise)"
                 +"\nThe average annual consumption is : "+avg+"\n"
                 +"The month of "+mini[1]+" has the minimum average
consumption of "+Float.parseFloat(mini[0])/1000
                 +"\nThe month of "+maxi[1]+" has the maximum average
consumption of "+Float.parseFloat(maxi[0])/1000);
        }
    }

    public static String findmin(int[] array){
        int minValue = array[0];
        int index=0;
        String min="";
        for(int i=1;i<array.length;i++){
        if(array[i] < minValue){
        minValue = array[i];
        index=i;
           }

        }
        switch(index)
        {
        case 0:min="January";break;
        case 1:min="February";break;
        case 2:min="March";break;
        case 3:min="April";break;
        case 4:min="May";break;
        case 5:min="June";break;
        case 6:min="July";break;
        case 7:min="August";break;
        case 8:min="September";break;
        case 9:min="October";break;
        case 10:min="November";break;
        case 11:min="December";break;


        }

       return minValue+" "+min ;
   }
    public static String findmax(int[] array){
        int maxValue = array[0];
        int index=0;
        String max="";
        for(int i=1;i<array.length;i++){
        if(array[i] > maxValue){
        maxValue = array[i];
```

```
        index=i;
           }

        }
        switch(index)
        {
        case 0:max="January";break;
        case 1:max="February";break;
        case 2:max="March";break;
        case 3:max="April";break;
        case 4:max="May";break;
        case 5:max="June";break;
        case 6:max="July";break;
        case 7:max="August";break;
        case 8:max="September";break;
        case 9:max="October";break;
        case 10:max="November";break;
        case 11:max="December";break;


        }

       return maxValue+" "+max ;
   }


}
```

data set : elect.txt

download dataset from :

https://github.com/parshva45/Big-Data-Analytics/blob/master/Mini%20Project/elect.txt

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf ec.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/ec/input/

$ hadoop fs -put -f elect.txt /user/hadoop/ec/input/

$ hadoop fs -ls /user/hadoop/ec/input/

$ hadoop jar ec.jar /user/hadoop/ec/input /user/hadoop/ec/output

$ hadoop fs -cat /user/hadoop/ec/output/part-r-00000
```

output :

```
Year : 1303 has the maximum annual consumption of 9319
Year : 1438 has the minimum annual consumption of 3475
The month of July has the minimum total consumption of 539950 (monthwise)
```

```
The month of February has the maximum total consumption of 558107
(monthwise)
The average annual consumption is : 6600.832
The month of July has the minimum average consumption of 539.95
The month of February has the maximum average consumption of 558.107
```

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

averageRatingMapper.java

```
package KPI_3;

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

// input data from   ****userAgeOccupationGenreRatingReducer


public class averageRatingMapper extends Mapper<Object,Text,Text,Text> {

     //data format => age::occupation::genre             rating
(tab delimited)

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

          String age_occupation_genre = tokens[0];

          String rating = tokens[1];

          String splitAgain[] = tokens[0].split("::");

          long age = Long.parseLong(splitAgain[0]);

          if(age >=18)                                   //age groups to
consider => 18+ only
          {

          context.write(new Text(age_occupation_genre), new
Text(rating));

          }

     }


}
```

averageRatingReducer.java

```
package KPI_3;
```

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



public class averageRatingReducer extends Reducer <Text,Text,Text,Text>{

     @Override
     public void reduce(Text key,Iterable<Text>values,Context
context)throws IOException,InterruptedException
     {
          //key                  value (ratings)
     //age::occupation::genre    [ 1, 4 ,2,3,5,5,5 .......]

     //one user watching multiple movies

          double avg = 0.0;
        double sum = 0.0;
        long count = 0;

          for(Text val:values)
          {
               String temp = val.toString();
              long rating = Long.parseLong(temp);

             sum+=rating;
              count++;


          }

          avg = sum/count;

          String average_rating = String.valueOf(avg);

          context.write(new Text(average_rating), new Text(key));

     }


}
```

dataReducer.java

```
package KPI_3;

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

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

```
     // here we are getting input from  ***movieDataMapper*** and
***ratingDataMapper***


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

      //key                    value
     // movie-id           [ Adventure|comedy_movies , 23:1_ratings ....
]

          String genre = null;

          //for a given movie_id there can be only one genre and
multiple users so
          //using list to store => age:occupation
          ArrayList<String> arr = new ArrayList<String>();


          for(Text val:values)
          {
               String []tokens = val.toString().split("_");

               if(tokens[1].equals("movie"))    //from movieDataMapper
               {
                    genre = tokens[0];      //we must know the genre
first to write the data

                }

               else if(tokens[1].equals("ratings"))  //from
ratingDataMapper
               {
                    arr.add(tokens[0]);


               }


          }

          for(String val:arr)
          {

               String []splitAgain = val.split(":");
               String user_id = splitAgain[0];
               String rating = splitAgain[1];

               context.write(new Text(user_id), new
Text(genre+"::"+rating));
          }

     }
}
```

movieDataMapper.java

```
package KPI_3;

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,Text,Text>{

	//data from movie.dat

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

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

		String movie_id = tokens[0];

		String genre = tokens[2].trim();

		context.write(new Text(movie_id), new Text(genre+"_movie"));
	}
}
```

ratingDataMapper.java

```
package KPI_3;

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,Text,Text> {

	//data from rating.dat

	//data format => UserID::MovieID::Rating::Timestamp

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

			String []tokens = value.toString().split("::");

			String user_id = tokens[0];

			String movie_id = tokens[1];
```

```
                String star_rating = tokens[2];

                context.write(new Text(movie_id), new
Text(user_id+":"+star_rating+"_ratings"));
          }
}
```

userAgeOccupationGenreRatingReducer.java

```
package KPI_3;

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

public class userAgeOccupationGenreRatingReducer extends
Reducer<Text,Text,Text,Text> {

     //key              values
     //user_id         [ genre::rating_file2   ,
age::occupation_file1 .....]


     @Override
     public void reduce(Text key,Iterable<Text> values,Context
context)throws IOException,InterruptedException
     {
          String age = null;
          String occupation = null;


          //for a user_id , there can be only one age::occupation and
multiple genres::rating
          //ArrayList to store => genre::rating
          ArrayList<String> arr2 = new ArrayList<String>();

          for(Text val:values)
          {
               String []tokens = val.toString().split("_");
               String file = tokens[1];



               if(file.equals("file1"))  //means data from
userDataMapper
               {
                    String []splitAgain = tokens[0].split("::");
                    age = splitAgain[0];
                    occupation = splitAgain[1];

               }

               else if(file.equals("file2"))  //means data from
userGenreRatingMapper
```

```
                {

                      arr2.add(tokens[0]);

                }


           }



           for(String val:arr2)
           {
                String []splitAgain2 = val.toString().split("::");
                String genre = splitAgain2[0];
                String rating = splitAgain2[1];


                context.write(new Text(age+"::"+occupation+"::"+genre),
new Text(rating));

           }

     }

}
```

userDataMapper.java

```
package KPI_3;

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


public class userDataMapper extends Mapper<Object,Text,Text,Text> {

     //data from user.dat

   //data format => UserID::Gender::Age::Occupation::Zip-code


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

           String user_id = tokens[0];

           String age = tokens[2];

           String occupation = tokens[3];
```

```
          context.write(new Text(user_id), new
Text(age+"::"+occupation+"_file1"));
                        //user_id as key         // age::occupation as
value

     }

}
```

userGenreRatingMapper.java

```
package KPI_3;

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

// input data from *****dataReducer*****


public class userGenreRatingMapper extends Mapper<Object,Text,Text,Text>
{

     //data format => user_id     genre::rating    (tab delimited)


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

          String []tokens = value.toString().split("\t");
          String user_id  = tokens[0];


          context.write(new Text(user_id), new
Text(tokens[1]+"_file2"));
                         //user_id             genre::rating

     }
}
```

datasets download :

https://github.com/srjsunny/Movie_Lens_Data-Analysis/tree/master/dataSet

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf mt.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/mt/inp1/

$ hadoop fs -mkdir -p /user/hadoop/mt/inp2/

$ hadoop fs -mkdir -p /user/hadoop/mt/inp3/
```

```
$ hadoop fs -put -f movies.dat /user/hadoop/mt/inp1/

$ hadoop fs -put -f ratings.dat /user/hadoop/mt/inp2/

$ hadoop fs -put -f users.dat /user/hadoop/mt/inp3/

$ hadoop jar mt.jar KPI_3.Driver3 /user/565/mt/inp1 /user/565/mt/inp2
/user/565/mt/out1 /user/565/mt/inp3 /user/565/mt/out2 /user/565/mt/ofi

$ hadoop fs -cat /user/hadoop/mt/out1/part-r-00000

$ hadoop fs -cat /user/hadoop/mt/out2/part-r-00000

$ hadoop fs -cat /user/hadoop/mt/ofi/part-r-00000
```

output :

3768 Crime::3
3083 Crime::5

5458 Crime::1

4404 Crime::3

hadoop9 Crime::3

2038 Crime::4

3095 Crime::1

3031 Crime::4

...

25::15::Action|Adventure|Comedy|Sci-Fi 4

25::15::Action|Thriller 3

25::15::Crime|Drama|Romance|Thriller 3

25::15::Action|Drama|War 2

25::15::Drama|Romance 3

25::15::Action|Sci-Fi|Thriller 5

...

3.0 56::9::Romance

5.0 56::9::Romance|Thriller

1.0 56::9::Sci-Fi

5.0 56::9::Sci-Fi|War

3.8 56::9::Thriller

3.6666666666666665 56::9::Western

10. XYZ.com is an online music website where users listen to various tracks, the data gets collected which is given below.
The data is coming in log files and looks like as shown below.

```
UserId | TrackId | Shared | Radio | Skip
111115 | 222 | 0 | 1 | 0
111113 | 225 | 1 | 0 | 0
111117 | 223 | 0 | 1 | 1
111115 | 225 | 1 | 0 | 0
```

Write a MapReduce program to get the following ▫
Number of unique listeners ▫
Number of times the track was shared with others ▫
Number of times the track was listened to on the radio ▫
Number of times the track was listened to in total ▫
Number of times the track was skipped on the radio

MusicTrack.java

```
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
//import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class MusicTrack
{
public static class MusicMapper extends Mapper<Object,Text,Text,Text>
{
    public void map(Object key,Text value,Context context) throws
IOException,InterruptedException
    {
        String[] tokens=value.toString().split("\\|");
        String trackid = /*"1";*/tokens[1];
        String others = tokens[0]+"\t"+tokens[2]+"\t"+tokens[3]+"\
t"+tokens[4];
        context.write(new Text(trackid),new Text(others));
    }
}

public static class MusicReduceer extends Reducer<Text,Text,Text,Text>
    {
    public void reduce(Text Key,Iterable<Text> value,Context context)
throws IOException,InterruptedException
        {

        Set<Integer> userIdSet = new HashSet<Integer>();
        int shared = 0;
        int radio =0;
        int skip= 0;
```

```
        int listen=0;

        for(Text val:value)

        {
                String[] valTokens = val.toString().split("\t");

                int sh = Integer.parseInt(valTokens[1]);
                int ra = Integer.parseInt(valTokens[2]);
                int sk = Integer.parseInt(valTokens[3]);

                shared = shared+sh;
                radio=radio+ra;
                skip=skip+sk;
                listen = shared + radio;

                int cus = Integer.parseInt(valTokens[0]);

                userIdSet.add(cus);

        }

        IntWritable size = new IntWritable(userIdSet.size());

    context.write(new Text(Key),new Text("customerId- "+size+"\
t"+"Shared- "+shared+"\t"+"Radio- "+radio+"\t"+"Skipped- "+skip+"\
t"+"Listen- "+listen));
    }

}
    public static void main(String args[]) throws Exception
        {
            Configuration conf=new Configuration();
            Job job=new Job(conf,"MusicTrack");
            job.setNumReduceTasks(1);
            job.setJarByClass(MusicTrack.class);
            job.setMapperClass(MusicMapper.class);

            job.setReducerClass(MusicReduceer.class);

            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.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,true);
            System.exit(job.waitForCompletion(true)?0:1);
        }
}
```

dataset : data.txt

```
UserId | TrackId | Shared | Radio | Skip
111115 | 222 | 0 | 1 | 0
111113 | 225 | 1 | 0 | 0
```

```
111117 | 223 | 0 | 1 | 1
111115 | 225 | 1 | 0 | 0
```

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf music.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/music/input/

$ hadoop fs -put -f data.txt /user/hadoop/music/input/

$ hadoop jar music.jar MusicTrack /user/hadoop/music/input
/user/hadoop/music/output

$ hadoop fs -cat /user/hadoop/music/output/part-r-00000
```

output :

```
17/08/05 15:25:16 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
225     1
521     1
```

11. Develop a MapReduce program to find the frequency of books published eachyear and find in which year maximum number of books were published usingthe following data.

Title Author Published year Author country Language No of pages

MainDriver.java

```
package com.mapreduce.classes;

import java.io.IOException;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;

public class MainDriver extends Configured implements Tool {


        static int printUsage() {
                System.out
                                .println("BookPubicationAnalysis [-m
<maps>] [-r <reduces>] <RatingssDetailInput> <RatingsOutput>
<BooksDetailsFile> <YOPOutput>");
```

```
                return 0;
        }

        @Override
        public int run(String[] args) throws IOException {
                return 0;
        }

        public static void main(String[] args) throws IOException {
                //first Job to map the Book, yop and ratings from ratings
Input file
                JobConf conf = new JobConf(MainDriver.class);
                conf.setJobName("BookPubicationAnalysisJOB1");

                conf.setOutputKeyClass(Text.class);
                conf.setOutputValueClass(Text.class);

                conf.setMapperClass(RatingsMapper.class);
                conf.setReducerClass(RatingsReducer.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);

                //second program to map books, yop and ratings from Books
Input file.
                JobConf conf2 = new JobConf(MainDriver.class);
                conf2.setJobName("BookPubicationAnalysisJOB2");

                conf2.setOutputKeyClass(Text.class);
                conf2.setOutputValueClass(Text.class);

                conf2.setMapperClass(YOPMapper.class);
                //conf2.setReducerClass(YOPReducer.class);

                conf2.setInputFormat(TextInputFormat.class);
                conf2.setOutputFormat(TextOutputFormat.class);

                FileInputFormat.setInputPaths(conf2, new Path(args[2]));
                FileOutputFormat.setOutputPath(conf2, new Path(args[3]));

                JobClient.runJob(conf2);
        }
}
```

RatingsMapper.java

```
package com.mapreduce.classes;

import java.io.IOException;

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

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class RatingsMapper extends MapReduceBase implements
                Mapper<LongWritable, Text, Text, Text> {
        private Text isbn;
        private Text rating;

        @Override
        public void map(LongWritable key, Text value,
                        OutputCollector<Text, Text> output, Reporter
reporter)
                        throws IOException {
                String[] rows = value.toString().split("\";\"");
                isbn = new Text(rows[1]);// second field is ISBN Number
of the book
                rating = new Text("0000" + "\t"
                                + rows[2].substring(0, rows[2].length() -
1));// third field

// is rating

// of the

// book
                output.collect(isbn, rating);
        }
}
```

RatingsReducer.java

```
package com.mapreduce.classes;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class RatingsReducer extends MapReduceBase implements
Reducer<Text, Text, Text, Text> {
@Override
public void reduce(Text key, Iterator<Text> values,
        OutputCollector<Text, Text> output, Reporter reporter)
        throws IOException {

Text redVal = new Text();
int sum = 0;
int count = 0;
int avgRatings = 0;
String yop = null;
```

```
String[] tempVal = null;

while (values.hasNext()) {
        tempVal = (values.next().toString()).split("\t");
        sum = sum + Integer.parseInt(tempVal[1]);
        count++;
}

avgRatings = Math.round(sum / count);
yop = tempVal[0];
redVal.set(yop + "\t" + avgRatings);
System.out.println("-----------> Reducer Values : " + redVal);
output.collect(key, redVal);
}
}
```

YOPMapper.java

```
package com.mapreduce.classes;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class YOPMapper extends MapReduceBase implements
                Mapper<LongWritable, Text, Text, Text> {
        private Text isbn;
        private Text yop;

        @Override
        public void map(LongWritable key, Text value,
                        OutputCollector<Text, Text> output, Reporter
reporter)
                        throws IOException {
                String[] rows = value.toString().split("\";\"");
                isbn = new Text(rows[0].substring(1));
                yop = new Text(rows[3].substring(0, rows[3].length()) +
"\t" + "00");
                output.collect(isbn, yop);
        }
}
```

YOPReducer.java

```
package com.mapreduce.classes;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
```

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

public class YOPReducer extends MapReduceBase implements
                Reducer<Text, Text, Text, Text> {
        @Override
        public void reduce(Text key, Iterator<Text> values,
                        OutputCollector<Text, Text> output, Reporter
reporter)
                        throws IOException {

                Text redVal = new Text();
                int sum = 0;
                int count = 0;
                int avgRatings = 0;
                String yop = null;
                String[] tempVal = null;

                while (values.hasNext()) {
                        tempVal = (values.next().toString()).split("\t");
                        sum = sum + Integer.parseInt(tempVal[1]);
                        count++;
                }

                avgRatings = Math.round(sum / count);
                yop = tempVal[0];
                redVal.set(yop + "\t" + avgRatings);
                System.out.println("-----------> Reducer Values : " +
redVal);
                output.collect(key, redVal);
        }
}
```

download dataset :

https://github.com/im-naren/BookPublicationAnalysis/tree/master/input

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf books.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/books/input/

$ hadoop fs -put -f books.csv /user/hadoop/books/input/

$ hadoop fs -put -f rating.csv /user/hadoop/books/input/


$ hadoop jar books.jar com/mapreduce/classes/MainDriver
/user/565/books/input/ratings.csv /user/565/books/output_ratings
/user/565/books/input/books.csv /user/565/books/ouput_books

$ hdfs dfs -cat /user/565/books/ouput_books/part-00000
```

```
$ hdfs dfs -cat /user/565/books/ouput_ratings/part-00000
```

output :

```
0002005018 2001  00
0060973129 1991  00
0195153448 2002  00
0374157065 1999  00
0393045218 1999  00
0399135782 1991  00
0425176428 2000  00
0440234743 1999  00
0671870432 1993  00
0679425608 1996  00
074322678X 2002  00
0771074670 1988  00
080652121X 2000  00
0887841740 2004  00
1552041778 1999  00
1558746218 1998  00
1567407781 1998  00
1575663937 1999  00
1881320189 1994  00


0060517794 0000  9
0155061224 0000  5
034545104X 0000  0
038550120X 0000  7
0425115801 0000  0
0446520802 0000  0
0449006522 0000  0
0451192001 0000  0
052165615X 0000  3
0521795028 0000  6
0553561618 0000  0
055356451X 0000  0
0600570967 0000  6
0609801279 0000  0
0786013990 0000  0
0786014512 0000  0
2080674722 0000  0
3257224281 0000  8
342310538  0000  10
```

12. 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

Average_age.java

```
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

   public class 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 )
throws IOException, 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);

      }

    }

    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));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

            @SuppressWarnings("deprecation")
         Job job = new Job(conf, "Averageage_survived");
            job.setJarByClass(Average_age.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(IntWritable.class);
         //  job.setNumReduceTasks(0);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
         Path out=new Path(args[1]);
         out.getFileSystem(conf).delete(out);
        job.waitForCompletion(true);
    }

  }
```

download dataset :

https://github.com/nttarun/Titanic-Data-Analysis/blob/master/TitanicData.txt

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf  Average_age.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/titanic/input/

$ hadoop fs -put -f TitanicData.txt /user/hadoop/titanic/input/

$ hadoop jar Average_age.jar Average_age /user/hadoop/titanic/input
/user/hadoop/titanic/output_Average_age

$ hadoop fs -cat /user/hadoop/titanic/output/_Average_age/part-r-00000
```

output

```
female     28
male       30
```

Average_fare.java

```
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

   public class Average_fare {

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

       private Text pclass = new Text();
       private FloatWritable fare = new FloatWritable();
       public void map(LongWritable key, Text value, Context context )
throws IOException, InterruptedException {
           String line = value.toString();
           String str[]=line.split(",");
           if(str.length>10){
            pclass.set(str[2]);
              if(str[9].matches("\\d+.+")){
                  float i=Float.parseFloat(str[9]);
                  fare.set(i);

              }
              context.write(pclass, fare);
     }


      }

    }

    public static class Reduce extends Reducer<Text,FloatWritable, Text,
FloatWritable> {

       public void reduce(Text key, Iterable<FloatWritable> values,
Context context)
         throws IOException, InterruptedException {
           float sum = 0;
           int l=0;
           for (FloatWritable val : values) {
               l+=1;
               sum += val.get();
           }
```

```
            sum=sum/l;
            context.write(key, new FloatWritable(sum));
         }
      }

      public static void main(String[] args) throws Exception {
         Configuration conf = new Configuration();

            @SuppressWarnings("deprecation")
          Job job = new Job(conf, "Averageage_survived");
            job.setJarByClass(Average_fare.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(FloatWritable.class);
          //  job.setNumReduceTasks(0);
         job.setOutputKeyClass(Text.class);
         job.setOutputValueClass(FloatWritable.class);

         job.setMapperClass(Map.class);
         job.setReducerClass(Reduce.class);

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

         FileInputFormat.addInputPath(job, new Path(args[0]));
         FileOutputFormat.setOutputPath(job, new Path(args[1]));
          Path out=new Path(args[1]);
          out.getFileSystem(conf).delete(out);
         job.waitForCompletion(true);
      }

   }
```

download dataset :

https://github.com/nttarun/Titanic-Data-Analysis/blob/master/TitanicData.txt

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf  Average_fare.jar *.class

$ hadoop fs -mkdir -p /user/hadoop/titanic/input/

$ hadoop fs -put -f TitanicData.txt /user/hadoop/titanic/input/

$ hadoop jar Average_fare.jar Average_fare /user/hadoop/titanic/input
/user/hadoop/titanic/output_Average_fare

$ hadoop fs -cat /user/hadoop/titanic/output/output_Average_fare/part-r-
00000
```

output

1	84.99818
2	21.659399
3	13.975503

13. 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

Uber.java

```
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 Uber {
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 block
e.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> values,
Context 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, "Uber");
job.setJarByClass(Uber.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);
}
}
```

download dataset :

https://raw.githubusercontent.com/shukladiwakar/Mapreduce-for-Uber-Data-Analysis/master/uberdata

commands for execution :

```
$ hadoop com.sun.tools.javac.Main *.java

$ jar cf Uber.jar *.class

$ hadoop fs -mkdir -p /user/565/uber/input/

$ hadoop fs -put -f uberdata /user/565/uber/input/

$ hadoop jar Uber.jar Uber /user/565/uber/input
/user/565/uber/output

$ hadoop fs -cat /user/565/uber/output/part-r-00000
```

output :

```
B02512 Fri    16435
B02512 Mon    11297
B02512 Sat    15026
B02512 Sun    10487
B02512 Thu    15809
B02512 Tue    12041
```

B02512 Wed 12691

14. Develop a program to calculate the maximum recorded temperature by yearwise for the weather dataset in Pig Latin

Download dataset :

```
wget https://raw.githubusercontent.com/pritambarlota/NCDC-weather-dataset-using-Hadoop-MapReduce-Pig-Hive/master/output1.txt
```

Running pig commands

```
$ pig -x local

grunt> records = LOAD '/home/hadoop/565/pig_mrt/output1.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, MAX(records.temperature);

grunt> DUMP max_temp;
```

output :

```
(1921,283)
(1922,278)
(1923,294)
(1924,294)
(1925,317)
(1926,261)
(1927,489)
(1928,178)
(1929,178)
(1930,228)
```

15. Write queries to sort and aggregate the data in a table using HiveQL.

Make dataset : emo_data.csv

```
123,Den,11000,Raphaely
124,Karen,2500,Colmenares
125,Susan,6500,Mavris
126,Jason,3300,Mallin
127,Alexis,4100,Bull
128,Kevin,3000,Feeney
129,Curtis,3100,Davies
130,John,2700,Seo
131,Stephen,3200,Stiles
132,Winston,3200,Taylor
133,James,2500,Marlow
134,Steven,2200,Markle
135,James,2400,Landry
136,Kevin,5800,Mourgos
137,Donald,2600,OConnell
138,Douglas,2600,Grant
139,Girard,2800,Geoni
140,Jean,3100,Fleaur
141,David,4800,Austin
```

Running HIVE commands:

```
$ hive

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

hive> LOAD DATA LOCAL INPATH
'/home/hadoop/565/hive_sql/emp_data.csv' INTO TABLE emp;

hive> SELECT * FROM emp;

hive> select * from emp sort by salary desc;
```

output :

```
123  Den       11000.0   Raphaely
125  Susan     6500.0    Mavris
136  Kevin     5800.0    Mourgos
141  David     4800.0    Austin
127  Alexis    4100.0    Bull
126  Jason     3300.0    Mallin
132  Winston   3200.0    Taylor
131  Stephen   3200.0    Stiles
```

```
129  Curtis    3100.0    Davies
140  Jean      3100.0    Fleaur
128  Kevin     3000.0    Feeney
139  Girard    2800.0    Geoni
130  John      2700.0    Seo
137  Donald    2600.0    OConnell
138  Douglas   2600.0    Grant
133  James     2500.0    Marlow
124  Karen     2500.0    Colmenares
135  James     2400.0    Landry
134  Steven    2200.0    Markle
```

16. Develop a Java application to find the maximum temperature using Spark.

Running Spark commands :

```
$ /usr/local/spark/bin/spark-shell --master "local[4]"

scala > import scala.collection.mutable.HashMap

scala > object MaxTempSpark{
     var data = HashMap(
          "Agra" -> 52,
          "Allahabad" -> 51,
          "Amritsar" -> 45,
          "Bhopal" -> 51,
          "Chandigarh" -> 47,
          "Dehradun" -> 43,
          "Indore" -> 50,
          "Lucknow" -> 58
     )
     print(data.max)
}

scala > MaxTempSpark
```

output :

```
(Lucknow,58)res0: MaxTempSpark.type = MaxTempSpark$@190c2bbf
```