2.1 Wordcount Program

2.2 WordSizeWordCount Program

2.3 WeatherData Program

2.4 Patent Program

2.5 MaxTemp Program

2.6 AverageSalary Program

2.7 De Identify HealthCare Program

2.8 Music Track Program

2.9 Telecom Call Data Record Program

2.10 Count Connected Component Program

2.10.1 Problem statement

You are given an adjacency list representation of an undirected graph. Your task is to write a MapReduce program that counts the number of connected components in the graph.

2.10.2 Input

The input consists of a list of vertices and their adjacent vertices. Each vertex is represented by a unique number listed on the leftmost side of the input. The adjacent vertices of each vertex are listed after it, separated by a space.

2.10.3 Output

Number of conntected component in graph.

Input	Output
0 9 1 4 9 2 7 3 5 8 4 1 5 3 6 7 2 8 3 9 0	4

CLOUDERA

Hadoop-MapReduce Lab

Map |Reduce |Driver

Sriram Balasubramanian

2016

CALIFORNIA, UNITED STATES OF AMERICA

Table of Contents

Hadoop - MapReduce Lab Assignment

Assignment 1 -Wordcount Program

Apply your MapReduce programming knowledge and write a MapReduce program to process a text file. You need to print the count of number of occurrences of each word in the text file.
The dataset for this problem is the text file **'wordcount'** available in your Lab

Problem statement

Let's understand the problem through a sample text file content:
"Hello everyone this is a sample dataset. You need to print the word count of particular words in this dataset."
Your MapReduce program should process this text file and should provide output as follows:

Output

Word	Word Count
a	1 (As the word 'a' occurred only once)
this	2 (As the word 'this' occurred twice)

Solution

Step 1. Project Creation

File->New->Java Project->project name: "**MR"** and then
Click **Finish** button

Step 2. Package Creation

Expand the project click **"src"**->right click->new package->give package name as "com.mr" and then
Click **Finish** button

Step 3. Class Creation

Right click com package->new class-> give class name as "**WordCount**" and then
Click **Finish** button

Step 4. Add External Jars

Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 5. Type the following MapReduce Program "WordCount"

```
package com.mr;
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.LongWritable;
import org.apache.hadoop.io.Text;
```

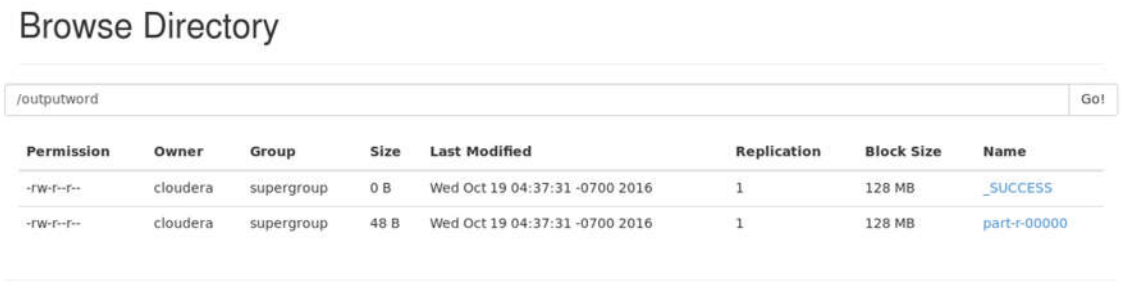
```
Import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable>
        {
                private final static IntWritable one=new IntWritable(1);
                private Text word=new Text();

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

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

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

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

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

        }
```

}

Step 6. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step 7. WordCount Execution:
[cloudera@quickstart ~]$ Hadoop fs –mkdir /sriMR

[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/srihadoop/MR/WordCount/WordCount.txt /user/cloudera/sriMR
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/WordCount/WordCount.jar com.mr.WordCount /user/cloudera/sriMR/inputword.txt /user/cloudera/sriMR/WordCountresult



Input file:

inputword.txt

hello welcome
welcome to big data
data is good

Debugging

Ctrl+Shift+O

Org.apache.hadoop.io.Text
Org.apache.hadoop.mapreduce.Job
Org.apache.hadoop.mapreduce.lib.output.FileOutputFormat
Org.apache.hadoop.mapreduce.lib.input.FileInputFormat
Org.apache.hadoop.mapreduce.lib.output.TextOutputFormat
Org.apache.hadoop.mapreduce.lib.input.TextInputFormat

Assignment 2 -WordSizeWordCount Program

Apply your MapReduce programming knowledge and write a MapReduce program to process two text files. You need to calculate the size of each word and count the number of words of that size in the text file.
The dataset for this problem is the text file **'alphabets'** available in your LMS.(http://www.gutenberg.org/files/4300/4300-0.txt)

Problem statement

Let's understand the problem through a sample text file content:

"Hello everyone this is a sample dataset. Calculate the word size and count the number of words of that size in this text file."

Your MapReduce program should process this text file and should provide output as follows:

Sample Output

Word Size	Word Count
1	1 (As the word of size 1 is: a)
2	4 (As the words of size 2 are: is, of, of, in)
3	3 (As the words of size 3 are: the, and, the)
4	6 (As the words of size 4 are: this, word, size, that, size)

Solution

Step 1. Class Creation

Right click com package->new class-> give class name as "**WordSizeWordCount**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)

Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 3. Type the following MapReduce Program "WordSizeWordCount"

```
package com.mr;
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.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.input.TextInputFormat;
```

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

public class WordSizeWordCount {

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

    //Defining a local variable count of type IntWritable
          private static IntWritable count ;

    //Defining a local variable word of type Text
      private Text word = new Text();
```

//Mapper

```
          /**
           * @method map
           * <p>This method takes the input as text data type and splits the input into words.
           * Now the length of each word in the input is determined and key value pair is made.
           * This key value pair is passed to reducer.
           * @method_arguments key, value, output, reporter
           * @return void
           */

       /*
       * (non-Javadoc)
       * @see org.apache.hadoop.mapred.Mapper#map(java.lang.Object, java.lang.Object,
org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)
       */

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

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

        //StringTokenizer is breaking the record (line) into words
        StringTokenizer tokenizer = new StringTokenizer(line);

        //iterating through all the words available in that line and forming the key value pair
        while (tokenizer.hasMoreTokens()) {

          String thisH = tokenizer.nextToken();

          //finding the length of each token(word)
          count= new IntWritable(thisH.length());
          word.set(thisH);

          //Sending to output collector which in turn passes the same to reducer
          //So in this case the output from mapper will be the length of a word and that word
          context.write(count,word);
        }
      }
  }
```

//Reducer

```
        public static class Reduce extends Reducer<IntWritable, Text, IntWritable, IntWritable> {

        /**
         * @method reduce
         * <p>This method takes the input as key and list of values pair from mapper, it does aggregation
         * based on keys and produces the final output.
         * @method_arguments key, values, output, reporter
         * @return void
         */

                /*
                 * (non-Javadoc)
                 * @see org.apache.hadoop.mapred.Reducer#reduce(java.lang.Object, java.util.Iterator,
org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)
                 */

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

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

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

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

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

  }
```

//Driver

```
  /**
   * @method main
   * <p>This method is used for setting all the configuration properties.
   * It acts as a driver for map reduce code.
   * @return void
   * @method_arguments args
   * @throws Exception
   */

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

        //reads the default configuration of cluster from the configuration xml files

        Configuration conf = new Configuration();

        //Initializing the job with the default configuration of the cluster
```

```
                Job = new Job(conf, "Wordsize");

                //Assigning the driver class name

                job.setJarByClass(WordSizeWordCount.class);

                //Defining the mapper class name

                job.setMapperClass(Map.class);

                //Defining the reducer class name

                job.setReducerClass(Reduce.class);

                //Defining the output key class for the mapper

                job.setMapOutputKeyClass(IntWritable.class);

                //Defining the output value class for the mapper

                job.setMapOutputValueClass(Text.class);

                //Defining the output key class for the final output i.e. from reducer

                job.setOutputKeyClass(IntWritable.class);

                //Defining the output value class for the final output i.e. from reduce

                job.setOutputValueClass(IntWritable.class);

                //Defining input Format class which is responsible to parse the dataset into a key value pair

                job.setInputFormatClass(TextInputFormat.class);

                //Defining output Format class which is responsible to parse the final key-value output from MR framework to
a text file into the hard disk

                job.setOutputFormatClass(TextOutputFormat.class);

                 //setting the second argument as a path in a path variable

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

                 //Configuring the input/output path from the filesystem into the job

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

                 //deleting the output path automatically from hdfs so that we don't have delete it explicitly

                  outputPath.getFileSystem(conf).delete(outputPath);

                  //exiting the job only if the flag value becomes false

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

}
```

Step 4. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step 5 WordSizeWordCount Execution:
[cloudera@quickstart ~]$ hadoop fs -put
/home/cloudera/Desktop/srihadoop/MR/WordSizeWordCount/WordSizeWordCount.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/WordSizeWordCount/WordSizeWordCount.jar
com.mr.WordSizeWordCount /user/cloudera/sriMR/WordSizeWordCount.txt /user/cloudera/sriMR/WordSizeWordCountresult

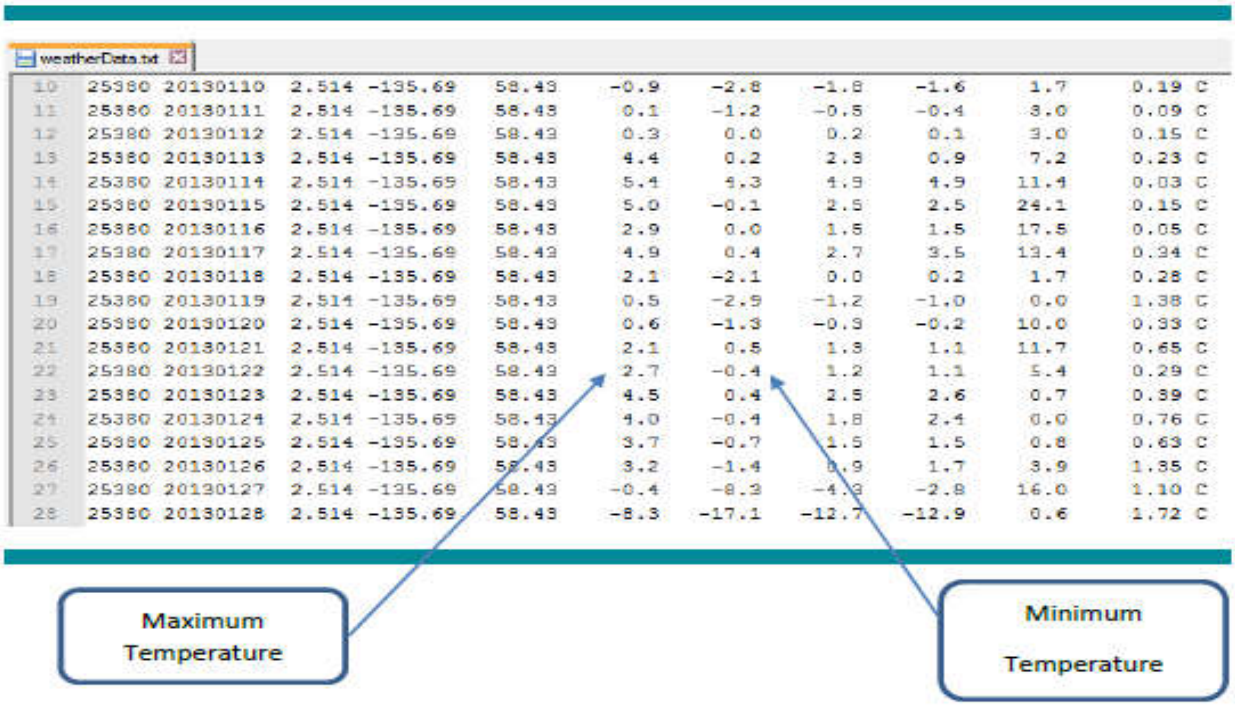
Assignment 3 - WeatherData Program

Apply your MapReduce programming knowledge and write a MapReduce program to process a dataset with temperature records. You need to find the Hot and Cold days in a year based on the maximum and minimum temperatures on those days. The dataset for this problem is the **'WeatherData'** records file available in your LMS. This dataset has been taken from National Climatic Data Center (NCDC) public datasets. You can download more datasets from this FTP site and review the README file to understand the available datasets. (ftp://ftp.ncdc.noaa.gov/pub/data/gsod/)

Problem statement

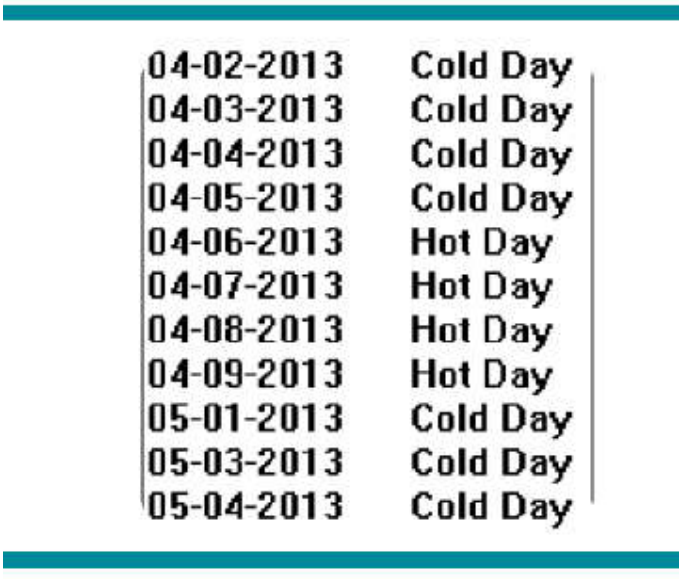
Let's understand the problem through a subset of records in the dataset as shown in the following figure:

FIGURE shows WEATHER RECORDS



Your task is to find out the dates with maximum temperature greater than 40 **(A Hot Day)** and minimum temperature lower than 10 **(A Cold Day)**. Here is the sample output:

FIGURE shows SAMPLE OUTPUT



You can review the solution in your Lab.

Solution

Step 1. Class Creation

Right click com package->new class-> give class name as "**WeatherData**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)

Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 3. Type the following MapReduce Program "WeatherData"

```
package com.mr;
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.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;


public class WeatherData {

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

@Override
public void map(LongWritable arg0, Text Value,
                    OutputCollector<Text, Text> output, Reporter arg3)
                    throws IOException {

          String line = Value.toString();

            // Example of Input
            //       Date                    Max    Min
            // 25380 20130101  2.514 -135.69  58.43    8.3    1.1    4.7    4.9    5.6    0.01 C    1.0    -0.1    0.4    97.3    36.0    69.4
-99.000 -99.000 -99.000 -99.000 -99.000 -9999.0 -9999.0 -9999.0 -9999.0 -9999.0
```

```
		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 > 40.0) {
			// Hot day
			output.collect(new Text("Hot Day " + date),
					new Text(String.valueOf(temp_Max)));
		}

		if (temp_Min < 10) {
			// Cold day
			output.collect(new Text("Cold Day " + date),
					new Text(String.valueOf(temp_Min)));
		}
}

}

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

@Override
public void reduce(Text Key, Iterator<Text> Values,OutputCollector<Text, Text> output, Reporter arg3) throws IOException {

		// Find Max temp yourself ?
		String temperature = Values.next().toString();
		output.collect(Key, new Text(temperature));
}

}

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

JobConf conf = new JobConf(WeatherData.class);
conf.setJobName("temp");

// Note:- As Mapper's output types are not default so we have to define the following properties.

conf.setMapOutputKeyClass(Text.class);
conf.setMapOutputValueClass(Text.class);

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

}

}
```

Step 4. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step 5WeatherData Execution:
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/ WeatherData.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/WeatherData/WeatherData.jar com.mr.WeatherData /user/cloudera/sriMR/WeatherData.txt /user/cloudera/sriMR/WeatherDataCountresult
hadoop jar /home/cloudera/Desktop/srihadoop/MR/WeatherData/WeatherData.jar com.mr.WeatherData /user/cloudera/sriMR/WeatherData1.txt /user/cloudera/sriMR/WeatherData1Countresult

Assignment 4 - Patent Program

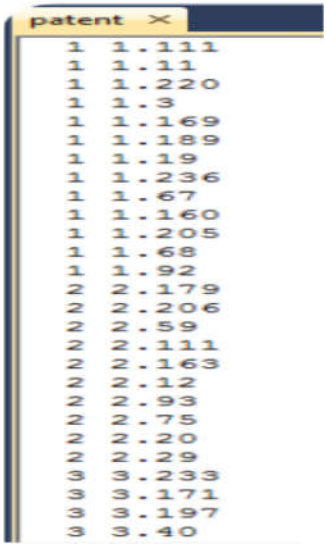
Apply your MapReduce programming knowledge and write a MapReduce program to process a dataset with patent records. You need to calculate the number of sub-patents associated with each patent.

The dataset for this problem is the **'patent'** records file available in your Lab.

Problem statement

Let's understand the problem through a subset of patent records as shown in the following figure:

FIGURE 1-1 PATENT RECORDS



Each patent has sub-patent ids associated with it. You need to calculate the number of sub-patent associated with each patent. Here is the sample output:

Sample Output

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

Your task in this assignment is to process the **'patent'** records using MapReduce program and count the number of associated sub-patents for each patent is in this dataset.

You can review the solution in your Lab.

Solution

Step 1. Class Creation
Right click com package->new class-> give class name as "**Patent**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)
Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 3. Type the following MapReduce Program "Patent"

```
package com.mr;
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.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Patent {
        /*
         * Map class is static and extends MapReduceBase and implements Mapper
         * interface having four hadoop generics type LongWritable, Text, Text, Text.
         */

        public static class Map extendsMapper<LongWritable, Text, Text, Text> {

//Mapper

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

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

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

    @Override
```

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

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

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

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

      while (tokenizer.hasMoreTokens()) {

        /*
         * The first token is going in jiten, second token in jiten1, third token in jiten,
         * fourth token in jiten1 and so on.
         */

        String jiten= tokenizer.nextToken();
        k.set(jiten);
        String jiten1= tokenizer.nextToken();
        v.set(jiten1);

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


  /*Reducer
   *
   * Reduce class is static and extends MapReduceBase and implements Reducer
   * interface having four hadoop generics type Text, Text, Text, IntWritable.
   */

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

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

        //Defining a local variable sum of type int

      int sum = 0;

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

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

      //Dumping the output in context object

      context.write(key, new IntWritable(sum));
    }
```

```
}


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

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

        //reads the default configuration of cluster from the configuration xml files

                Configuration conf = new Configuration();

                //Initializing the job with the default configuration of the cluster


                Job = new Job(conf, "patent");

                //Assigning the driver class name

                job.setJarByClass(Patent.class);

                //Defining the mapper class name

                job.setMapperClass(Map.class);

                //Defining the reducer class name

                job.setReducerClass(Reduce.class);

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

                //Defining the output key class for the final output i.e. from reducer

                job.setOutputKeyClass(Text.class);

                //Defining the output value class for the final output i.e. from reducer

                job.setOutputValueClass(IntWritable.class);

                //Defining the output key class for the final output i.e. from reducer

                job.setOutputKeyClass(Text.class);

                //Defining the output value class for the final output i.e. from reducer

                job.setOutputValueClass(Text.class);

                //Defining input Format class which is responsible to parse the dataset into a key value pair

                job.setInputFormatClass(TextInputFormat.class);
```

```
//Defining output Format class which is responsible to parse the final key-value output from MR framework to a text file into the hard disk

                job.setOutputFormatClass(TextOutputFormat.class);

    //setting the second argument as a path in a path variable

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

    //Configuring the input/output path from the filesystem into the job

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

                //deleting the output path automatically from hdfs so that we don't have delete it explicitly

                outputPath.getFileSystem(conf).delete(outputPath);

                //exiting the job only if the flag value becomes false

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

  }

}
```

Step 4. Export JAR file creation:

Right click src->Export->Java->JAR File->click Next button

Step 5 Patent Execution:

```
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/Patent.txt /user/cloudera/sriMR
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/Patent/Patent.jar com.mr.Patent
/user/cloudera/sriMR/Patent.txt /user/cloudera/sriMR/Patentresult
```

Assignment 5 - MaxTemp Program

Apply your MapReduce programming knowledge and write a MapReduce program to process a dataset with multiple temperatures for a year. You need to process the dataset to find out the maximum temperature for each year in the dataset. The dataset for this problem is the text file **'Temperature'** available in your Lab. (ftp://ftp.ncdc.noaa.gov/pub/data/noaa/)

Problem statement

Let's understand the problem through a subset of temperature records as shown in the following figure:

```
1900 36
1900 29
1901 32
1901 40
1901 29
1901 48
1901 16
1901 11
1901 21
1901 6
1901 22
1902 49
1902 49
```

In this data set, the first field represents the year and the second field represents the temperature in that year. As the temperature will not be constant throughout the year, each year has multiple temperatures listed in the dataset. You need to process the dataset and find the maximum temperature during a year. Here is the sample

Output: **Sample Output**

Year	Maximum Temperature
1900	36
1901	48
1902	49

Solution

Step 1. Class Creation

Right click com package->new class-> give class name as "**MaxTemp**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)

Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 3. Type the following MapReduce Program "MaxTemp"

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


public class MaxTemp {

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

        /**
         * @method map
         * <p>This method takes the input as text data type and tokenizes input
         * by taking whitespace as delimiter. The first token goes year and second token is temperature,
         * this is repeated till last token. Now key value pair is made and passed to reducer.
         * @method_arguments key, value, output, reporter
         * @return void
         */

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

        /*
         * (non-Javadoc)
         * @see org.apache.hadoop.mapred.Mapper#map(java.lang.Object, java.lang.Object,
org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)
         */

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

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

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

            //Iterating through all the tokens and forming the key value pair
            while (tokenizer.hasMoreTokens()) {

            //The first token is going in year variable of type string
            String year= tokenizer.nextToken();
            k.set(year);

            //Takes next token and removes all the whitespaces around it and then stores it in the string variable called temp
            String temp= tokenizer.nextToken().trim();

            //Converts string temp into integer v
            int v = Integer.parseInt(temp);

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

  //Reducer

  /**
   * @author sriram!
   * @interface Reducer
   * <p>Reduce class is static and extends MapReduceBase and implements Reducer
   * interface having four hadoop generics type Text, IntWritable, Text, IntWritable.
   */

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

            /**
             * @method reduce
             * <p>This method takes the input as key and list of values pair from mapper, it does aggregation
             * based on keys and produces the final output.
             * @method_arguments key, values, output, reporter
             * @return void
             */

             /*
              * (non-Javadoc)
              * @see org.apache.hadoop.mapred.Reducer#reduce(java.lang.Object, java.util.Iterator,
org.apache.hadoop.mapred.OutputCollector, org.apache.hadoop.mapred.Reporter)
              */

      @Override
      public void reduce(Text key, Iterable<IntWritable> values, Context context)
                        throws IOException, InterruptedException {
            /*
             * Iterates through all the values available with a key and if the integer variable temperature
             * is greater than maxtemp, then it becomes the maxtemp
             */

       //Defining a local variable maxtemp of type int
            int maxtemp=0;
          for(IntWritable it : values) {

          //Defining a local variable temperature of type int which is taking all the temperature
```

```
        int temperature= it.get();
          if(maxtemp<temperature)
          {
                    maxtemp =temperature;
          }
        }

        //Finally the output is collected as the year and maximum temperature corresponding to that year
        context.write(key, new IntWritable(maxtemp));
      }

  }

//Driver


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

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

          //reads the default configuration of cluster from the configuration xml files

                    Configuration conf = new Configuration();

                    //Initializing the job with the default configuration of the cluster


                    Job job = new Job(conf, "MaxTemp");

                    //Assigning the driver class name

                    job.setJarByClass(MaxTemp.class);

                    //Defining the mapper class name

                    job.setMapperClass(Map.class);

                    //Defining the reducer class name

                    job.setReducerClass(Reduce.class);

                    //Defining the output key class for the final output i.e. from reducer

                    job.setOutputKeyClass(Text.class);

                    //Defining the output value class for the final output i.e. from reducer

                    job.setOutputValueClass(IntWritable.class);

                    //Defining input Format class which is responsible to parse the dataset into a key value pair

                    job.setInputFormatClass(TextInputFormat.class);
```

```
//Defining output Format class which is responsible to parse the final key-value output from MR framework to a text file into the hard disk

                job.setOutputFormatClass(TextOutputFormat.class);

    //setting the second argument as a path in a path variable

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

    //Configuring the input/output path from the filesystem into the job

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

                //deleting the output path automatically from hdfs so that we don't have delete it explicitly

                outputPath.getFileSystem(conf).delete(outputPath);

                //exiting the job only if the flag value becomes false

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


  }
}
```

Step 4. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step 5 MaxTemp Execution:

```
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/MaxTemp.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/MaxTemp/MaxTemp.jar com.mr.MaxTemp /user/cloudera/sriMR/ MaxTemp.txt /user/cloudera/sriMR/MaxTempresult
```

Assignment 6 - AverageSalary Program

Problem statement

Calculate the average salary in the department.

Solution

Step 1. Class Creation
Right click com package->new class-> give class name as "**AverageSalary**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)
Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 3. Type the following MapReduce Program "AverageSalary"

```
package com.mr;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
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 AverageSalary
{

        public static class avgMapper extends Mapper<Object,Text,Text,FloatWritable>
        {
        private Text dept_id=new Text();
        private FloatWritable salary = new FloatWritable();
        public void map(Object key, Text value,Context context)throws IOException, InterruptedException{

                        String values[]=value.toString().split("\t");
                        dept_id.set(values[0]);
                        salary.set(Float.parseFloat(values[2]));
                        context.write(dept_id,salary);
                }
        }
```

```
	public static class avgReducer extends Reducer<Text,FloatWritable,Text,FloatWritable>{

		private FloatWritable result = new FloatWritable();
		public void reduce(Text key, Iterable<FloatWritable>values, Context context)throws IOException,
InterruptedException{
			float sum=0;
			float count =0;
			for(FloatWritable val: values){
				sum+=val.get();
				count++;

			}

			result.set(sum/count);
			context.write(key,result);
		}
	}

	public static void main(String[]args)throws Exception{
		Configuration conf = new Configuration();
		Job job=new Job(conf,"averagesal");
		job.setJarByClass(AverageSalary.class);
		job.setMapperClass(avgMapper.class);
		job.setCombinerClass(avgReducer.class);
		job.setReducerClass(avgReducer.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(FloatWritable.class);
		Path p=new Path(args[0]);
		Path p1=new Path(args[1]);

		FileInputFormat.addInputPath(job,p);
		FileOutputFormat.setOutputPath(job,p1);
		job.waitForCompletion(true);


	}
}
```

Step 4. Export JAR file creation:

Right click src->Export->Java->JAR File->click Next button

Step 5 AverageSalary Execution:

```
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/MaxTemp.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/AverageSalary/AverageSalary.jar
com.mr.AverageSalary /user/cloudera/sriMR/averagesalary.txt /user/cloudera/sriMR/AverageSalaryresult2
```

Assignment 7 - De Identify HealthCare Program

Problem statement

Populate the healthcare dataset with the following fields –

PatientID, Name, DOB, Phone Number, Email_Address, SSN, Gender, Disease, weight

PatientID	Name	DOB	Phone Number	Email_Address	SSN	Gender	Disease	weight
11111	bbb1	12/10/1950	1.23E+09	bbb1@xxx.com	1.11E+09	M	Diabetes	78
11112	bbb2	12/10/1984	1.23E+09	bbb2@xxx.com	1.11E+09	F	PCOS	67
11113	bbb3	712/11/1940	1.23E+09	bbb3@xxx.com	1.11E+09	M	Fever	90
11114	bbb4	12/12/1950	1.23E+09	bbb4@xxx.com	1.11E+09	F	Cold	88
11115	bbb5	12/13/1960	1.23E+09	bbb5@xxx.com	1.11E+09	M	Blood Pressure	76
11116	bbb6	12/14/1970	1.23E+09	bbb6@xxx.com	1.11E+09	F	Malaria	84

Solution

Step 1. Class Creation

Right click com package->new class-> give class name as "**DeIdentifyData**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)

Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 3. Type the following MapReduce Program "DeIdentifyData" (Program Works from JDK 1.8)

```
package com.mr;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.StringTokenizer;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;

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

```
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.log4j.Logger;

public class DeIdentifyData
{
static Logger = Logger.getLogger(DeIdentifyData.class.getName());
	public static Integer[] encryptCol={2,3,4,5,6,8};
	private static byte[] key1 = new String("samplekey1234567").getBytes();

	public static class Map extends Mapper<Object, Text, NullWritable, Text> {

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

		//value = PatientID, Name,DOB,Phone Number,Email_Address,SSN,Gender,Disease,weight

			StringTokenizer itr = new StringTokenizer(value.toString(),",");
			List<Integer> list=new ArrayList<Integer>();
			Collections.addAll(list, encryptCol);
			//list=2,3,4,5,6,8
			System.out.println("Mapper :: one :"+value);
			String  newStr="";

			int counter=1;

			while (itr.hasMoreTokens()) {
				String token=itr.nextToken();
				System.out.println("token"+token);
				System.out.println("i="+counter);
				if(list.contains(counter))
				{
					if(newStr.length()>0)
						newStr+=",";

					newStr+=encrypt(token, key1);

				}
				else
				{
					if(newStr.length()>0)
						newStr+=",";
					newStr+=token;
				}
				counter=counter+1;
			}
			context.write(NullWritable.get(), new Text(newStr.toString()));

		}
	}

	public static void main(String[] args) throws Exception {
				if (args.length != 2) {
	      System.out.println("usage: [input] [output]");
	      System.exit(-1);
	    }
```

```
        Job = Job.getInstance(new Configuration());

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

        job.setMapperClass(Map.class);

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

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

        job.setJarByClass(DeIdentifyData.class);
        job.waitForCompletion(true);

    }


    public static String encrypt(String strToEncrypt, byte[] key)
    {
        try
        {
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            //cipher.init(Cipher.DECRYPT_MODE, secretKey);
            String encryptedString = Base64.encodeBase64String(cipher.doFinal(strToEncrypt.getBytes()));
            //String decrypted = new String(cipher.doFinal(Base64.decodeBase64(strToEncrypt)));

            return encryptedString.trim();
            //return decrypted;
        }
        catch (Exception e)
        {
            logger.error("Error while encrypting", e);
        }
        return null;

    }


}
```

Step 4. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step 5 DeIdentify Execution:
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/MaxTemp.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/DeIdentify/DeIdentify.jar com.mr. DeIdentify /user/cloudera/sriMR/averagesalary.txt /user/cloudera/sriMR/DeIdentifyresult

Assignment 8 - Music Track Program

Problem Statement

XYZ.com is an online music website where users listen to various tracks, the data gets collected like shown below. Write a map reduce program to get following stats

- 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

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

Solution

First we are going to solve the first problem that is finding out unique listeners per track.

First of all we need to understand the data, here the first column is UserId and the second one is Track Id. So we need to write a mapper class which would emit trackId and userIds and intermediate key value pairs. so make it simple to remember the data sequence, let's create a constants class as shown below

```
package com.mr;
public class LastFMConstants {

   public static final int USER_ID = 0;
   public static final int TRACK_ID = 1;
   public static final int IS_SHARED = 2;
   public static final int RADIO = 3;
   public static final int IS_SKIPPED = 4;

}
```

Now, let's create the mapper class which would emit intermediate key value pairs as (TrackId, UserId) as shown below

Step 1. Class Creation

Right click com package->new class-> give class name as "**UniqueListener**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)

Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 3. Type the following MapReduce Program "UniqueListener"

```
public static class UniqueListenersMapper extends Mapper< Object , Text, IntWritable, IntWritable > {
      IntWritable trackId = new IntWritable();
      IntWritable userId = new IntWritable();

public void map(Object key, Text value, Mapper< Object , Text, IntWritable, IntWritable > .Context context)
      throws IOException, InterruptedException {

   String[] parts = value.toString().split("[|]");
   trackId.set(Integer.parseInt(parts[LastFMConstants.TRACK_ID]));
   userId.set(Integer.parseInt(parts[LastFMConstants.USER_ID]));
      if (parts.length == 5) {
      context.write(trackId, userId);
   } else {
      // add counter for invalid records
      context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L);
   }
   }
}
```

You would have also noticed that we are using a counter here named INVALID_RECORD_COUNT , to count if there are any invalid records which are not coming the expected format. Remember, if we don't do this then in case of invalid records, our program might fail.

Now let's write a Reducer class to aggregate the results. Here we simply can not use sum reducer as the records we are getting are not unique and we have to count only unique users. Here is how the code would look like

```
public static class UniqueListenersReducer extends Reducer< IntWritable , IntWritable, IntWritable, IntWritable> {

public void reduce(IntWritable trackId,Iterable< IntWritable > userIds,Reducer< IntWritable , IntWritable,
IntWritable,IntWritable>.Context context)  throws IOException, InterruptedException {

      Set< Integer > userIdSet = new HashSet< Integer >();
      for (IntWritable userId : userIds) {
      userIdSet.add(userId.get());
      }

      IntWritable size = new IntWritable(userIdSet.size());
      context.write(trackId, size);
   }
}
```

Here we are using Set to eliminate duplicate userIds. Now we can take look at the Driver class

```
public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      if (args.length != 2) {
         System.err.println("Usage: uniquelisteners < in >< out >");
         System.exit(2);
      }
      Job job = new Job(conf, "Unique listeners per track");
      job.setJarByClass(UniqueListeners.class);
      job.setMapperClass(UniqueListenersMapper.class);
      job.setReducerClass(UniqueListenersReducer.class);
      job.setOutputKeyClass(IntWritable.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);
    org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
    System.out.println("No. of Invalid Records :"
          + counters.findCounter(COUNTERS.INVALID_RECORD_COUNT).getValue());
  }
```

Step 4. Export JAR file creation:

Right click src->Export->Java->JAR File->click Next button

Step Music Track Execution:

[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/UniqueListener.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/UniqueListener/UniqueListener.jar com.mr.UniqueListener/user/cloudera/sriMR/UniqueListener.txt /user/cloudera/sriMR/UniqueListenerresult

Assignment 9 - Telecom Call Data Record Program

Problem Statement

We are going to solve a very useful problem called Call Data Record (CDR) Analytics.

A Telecom company keeps records for its subscribers in specific format.

Consider following format

FromPhoneNumber|ToPhoneNumber|CallStartTime|CallEndTime|STDFlag

Now we have to write a map reduce code to find out all phone numbers who are making more than 60 mins of STD calls. Here if STD Flag is 1 that means it was as STD Call. STD is call is call which is made outside of your state or long distance calls. By identifying such subscribers, Telcom Company wants to offer them STD (Long Distance) Pack which would efficient for them instead spending more money without that package. The data is coming in log files and looks like as shown below.

FromPhoneNumber|ToPhoneNumber|CallStartTime|CallEndTime|STDFlag

9665128505|8983006310|2015-03-01 07:08:10|2015-03-01 08:12:15|0
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 09:12:15|1
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|0
9665128506|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
9665128507|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1

Solution

First of all we need to understand the data, depending upon the output we are expecting, we need to write a mapper class which would emit FromPhoneNumber and Duration of STD Call intermediate key value pairs. To make it simple to remember the data sequence, let's create a constants class as shown below

```
package com.mr;
public class CDRConstants {

   public static int fromPhoneNumber = 0;
   public static int toPhoneNumber = 1;
   public static int callStartTime = 2;
   public static int callEndTime = 3;
   public static int STDFlag = 4;

}
```

Now, let's create the mapper class which would emit intermediate key value pairs as (FromPhoneNumber, Duration), here we would also need to use our Java skills to calculate duration (CallEndTime- CallStartTime). We are also making some manipulations to get duration in minutes

Step 1. Class Creation
Right click com package->new class-> give class name as "**CallDataRecord**" and then
Click **Finish** button

Step 2. Add External Jars (Added Already)
Add JARS file:
Right click "**src**"->**build path->configure build path-> click Libraries pane->add external jars->file system->**

Step 3. Type the following MapReduce Program "CallDataRecord"

```
public static class TokenizerMapper extendsMapper< Object , Text, Text, LongWritable> {

    Text phoneNumber = new Text();
    LongWritable durationInMinutes = new LongWritable();

    public void map(Object key, Text value,  Mapper< Object , Text, Text, LongWritable>.Context context)
          throws IOException, InterruptedException {
       String[] parts = value.toString().split("[|]");
       if (parts[CDRConstants.STDFlag].equalsIgnoreCase("1")) {

          phoneNumber.set(parts[CDRConstants.fromPhoneNumber]);
          String callEndTime = parts[CDRConstants.callEndTime];
          String callStartTime = parts[CDRConstants.callStartTime];
          long duration = toMillis(callEndTime) - toMillis(callStartTime);
          durationInMinutes.set(duration / (1000 * 60));
          context.write(phoneNumber, durationInMinutes);
       }
    }

    private long toMillis(String date) {

       SimpleDateFormat format = new SimpleDateFormat( "yyyy-MM-dd HH:mm:ss");
       Date dateFrm = null;
       try {
          dateFrm = format.parse(date);

       } catch (ParseException e) {

          e.printStackTrace();
       }
       return dateFrm.getTime();
    }
 }
```

You can also use counters in case you are not sure if the data you are receiving is correct or no like we did in previous tutorial.

Now that we have already done majority of things in Mapper Class itself, here a reduce would be a simple Sum Reducer. Here is how the code would look like

```
public static class SumReducer extends Reducer< Text , LongWritable, Text, LongWritable> {

private LongWritable result = new LongWritable();

public void reduce(Text key, Iterable< LongWritable> values, Reducer< Text , LongWritable, Text, LongWritable>.Context
context)
        throws IOException, InterruptedException {
      long sum = 0;
      for (LongWritable val : values) {
        sum += val.get();
      }
      this.result.set(sum);
      if (sum >= 60) {
        context.write(key, this.result);
      }

    }
  }
```

Now we can take look at the Driver class

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    if (args.length != 2) {
      System.err.println("Usage: stdsubscriber < in>< out>");
      System.exit(2);
    }
    Job = new Job(conf, "STD Subscribers");
    job.setJarByClass(STDSubscribers.class);

    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(SumReducer.class);
    job.setReducerClass(SumReducer.class);

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

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

  }
```

Step 4. Export JAR file creation:
Right click src->Export->Java->JAR File->click Next button

Step 5 CallDataRecord Execution:
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/ **CallDataRecord**.txt /user/cloudera/sriMR

[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/srihadoop/MR/ **CallDataRecord**/ **CallDataRecord**.jar com.mr.
CallDataRecord /user/cloudera/sriMR/ **CallDataRecord**.txt /user/cloudera/sriMR/ **CallDataRecord**result