UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Experiment No. 2.2

Student Name: Ravi **UID: 22MCC20032**
Branch: MCA - CCD **Section/Group: 22MCD-1/ Grp A**
Semester: IV **Date of Performance: 27th Feb 2024**
Subject Name: Big Data & Analytics Lab **Subject Code: 22CAH-782**

1. Aim/Overview of the practical:

Weather Report POC-Map Reduce Program to analyse time-temperature statistics and generate report with max/min temperature.

2. Code/Steps for practical:

Step-1. Write a Mapper
A Mapper overrides the ―map‖ function from the Class "org.apache.hadoop.mapreduce.Mapper" which provides <key, value> pairs as the input. A Mapper implementation may output <key,value> pairs using the provided Context .

Input value of the WordCount Map task will be a line of text from the input data file and the key would be the line number <line_number, line_of_text> . Map task outputs <word, one> for each word in the line of text.

Pseudo-code void
Map (key, value){
for each max_temp x in value: output.collect(x, 1);
}
void Map (key, value){
for each min_temp x in value: output.collect(x, 1); }

Step-2 Write a Reducer
A Reducer collects the intermediate <key,value> output from multiple map tasks and assemble a single result. Here, the WordCount program will sum up the occurrence of each word to pairs as <word, occurrence>.

Pseudo-code void Reduce (max_temp, <list of value>){ for each x in
<list of value>: sum+=x; final_output.collect(max_temp, sum);
} void Reduce (min_temp, <list of value>){ for each x in <list of
value>: sum+=x; final_output.collect(min_temp, sum); }

Step- 3 Write Driver

The Driver program configures and run the MapReduce job. We use the main program to perform basic configurations such as: Job Name: name of this Job
Executable (Jar) Class: the main executable class. For here, WordCount. Mapper Class: class which overrides the "map" function. For here, Map. Reducer: class which override the "reduce" function. For here, Reduce. Output Key: type of output key. For here, Text. Output Value: type of output value. For here, IntWritable.

File Input Path File Output Path

Here is the java code for map reduce:

```
import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


                        /**
                        * @author devinline
                        */
      public class CalculateMaxAndMinTemeratureWithTime {       public
static String calOutputName = "California";       public static String
nyOutputName = "Newyork";   public static String njOutputName = "Newjersy";
      public static String ausOutputName = "Austin";     public static
String bosOutputName = "Boston";   public static String balOutputName =
"Baltimore";        public static class WhetherForcastMapper extends
Mapper<Object, Text, Text, Text> {


       public void map(Object keyOffset, Text dayReport, Context con)      throws
IOException, InterruptedException {         StringTokenizer strTokens = new
StringTokenizer(        dayReport.toString(), "\t");      int counter = 0;
                         Float currnetTemp = null;
                         Float minTemp = Float.MAX_VALUE;
                         Float maxTemp = Float.MIN_VALUE;
                         String date = null;
                         String currentTime = null;
                         String minTempANDTime = null;
                         String maxTempANDTime = null;
```

```
        while (strTokens.hasMoreElements()) {        if
(counter == 0) {         date = strTokens.nextToken();
                          } else {
          if (counter % 2 == 1) {       currentTime =
strTokens.nextToken();
                           } else {
           currnetTemp = Float.parseFloat(strTokens.nextToken());           if
(minTemp > currnetTemp) {         minTemp = currnetTemp;
                             minTempANDTime = minTemp + "AND" + currentTime;
                            }
           if (maxTemp < currnetTemp) {         maxTemp =
currnetTemp;
                             maxTempANDTime = maxTemp + "AND" + currentTime;
                            }
                           }
                          }
                          counter++;
                         }
                         // Write to context - MinTemp, MaxTemp and corresponding time
                         Text temp = new Text();
                         temp.set(maxTempANDTime);
                       Text dateText = new Text();
                       dateText.set(date);   try {
                       con.write(dateText, temp);
                             } catch (Exception e)
                       {      e.printStackTrace();
                         }


        temp.set(minTempANDTime);    dateText.set(date);
con.write(dateText, temp);

                        }
                       }



                       public static class WhetherForcastReducer extends
        Reducer<Text, Text, Text, Text> {         MultipleOutputs<Text, Text>
mos;



       public void setup(Context context) {         mos = new
MultipleOutputs<Text, Text>(context);
                        }



       public void reduce(Text key, Iterable<Text> values, Context context)       throws
IOException, InterruptedException {
                         int counter = 0;
                         String reducerInputStr[] = null;
                         String f1Time = "";
```

```
                        String f2Time = "";
        String f1 = "", f2 = "";     Text result = new
Text();        for (Text value : values) {

                         if (counter == 0) {
          reducerInputStr = value.toString().split("AND");          f1 =
reducerInputStr[0];             f1Time = reducerInputStr[1];
                         }



                         else {
                          reducerInputStr =
                       value.toString().split("AND");     f2 =
                       reducerInputStr[0];     f2Time =
                       reducerInputStr[1];
                         }



                         counter = counter + 1;
                        }
                        if (Float.parseFloat(f1) > Float.parseFloat(f2)) {

                         result = new Text("Time: " + f2Time + " MinTemp: " + f2 + "\t"
                           + "Time: " + f1Time + " MaxTemp: " + f1);
                        } else {


                         result = new Text("Time: " + f1Time + " MinTemp: " + f1 + "\t"
                           + "Time: " + f2Time + " MaxTemp: " + f2);
                        }
                        String fileName = "";
        if (key.toString().substring(0, 2).equals("CA")) {         fileName =
CalculateMaxAndMinTemeratureTime.calOutputName;     } else if
(key.toString().substring(0, 2).equals("NY")) {      fileName =
CalculateMaxAndMinTemeratureTime.nyOutputName;      } else if
(key.toString().substring(0, 2).equals("NJ")) {      fileName =
CalculateMaxAndMinTemeratureTime.njOutputName;      } else if
(key.toString().substring(0, 3).equals("AUS")) {            fileName =
CalculateMaxAndMinTemeratureTime.ausOutputName;     } else if
(key.toString().substring(0, 3).equals("BOS")) {            fileName =
CalculateMaxAndMinTemeratureTime.bosOutputName;     } else if
(key.toString().substring(0, 3).equals("BAL")) {            fileName =
CalculateMaxAndMinTemeratureTime.balOutputName;
                        }
        String strArr[] = key.toString().split("_");       key.set(strArr[1]); //Key
is date value         mos.write(fileName, key, result);
                       }



                        @Override
                        public void cleanup(Context context) throws IOException,
         InterruptedException {      mos.close();
```

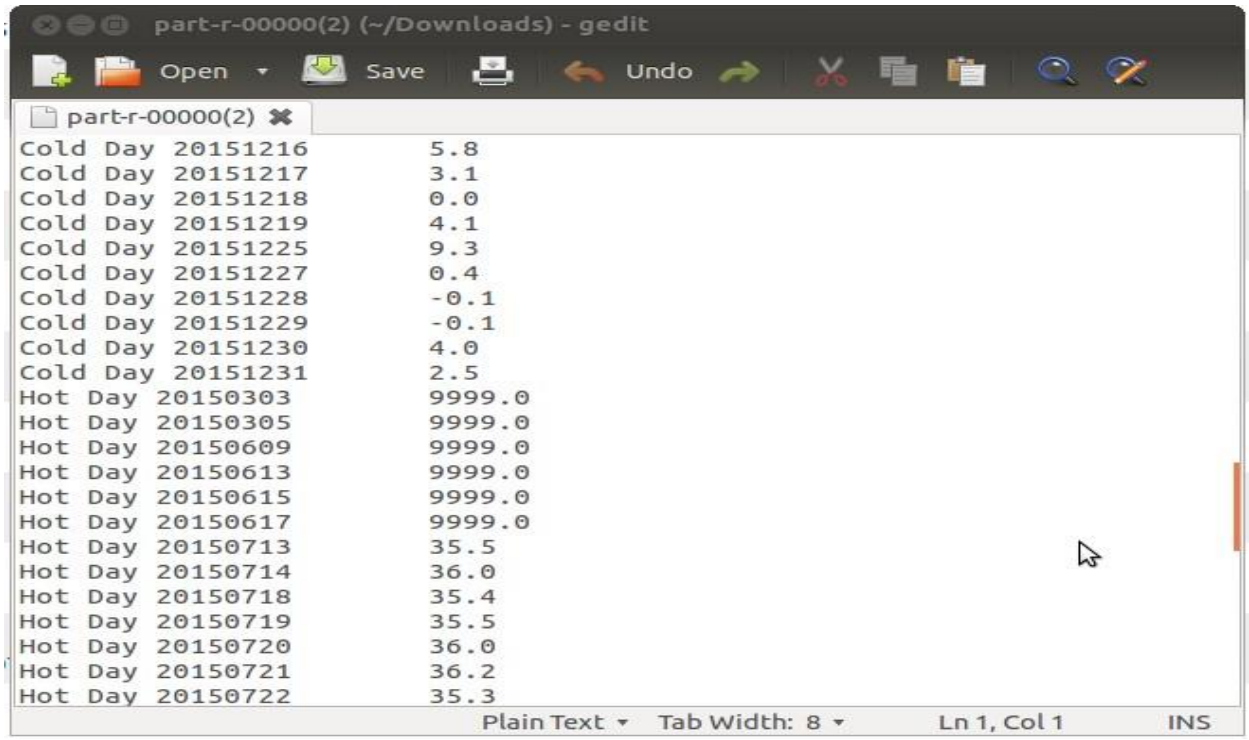
```
}
}

public static void main(String[] args) throws IOException,
  ClassNotFoundException, InterruptedException {
 Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "Wheather Statistics of USA");
job.setJarByClass(CalculateMaxAndMinTemeratureWithTime.class);
job.setMapperClass(WhetherForcastMapper.class);     job.setReducerClass(WhetherForcastReducer.class);

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

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

MultipleOutputs.addNamedOutput(job, calOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, nyOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, njOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, bosOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, ausOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, balOutputName,          TextOutputFormat.class,
Text.class, Text.class);

// FileInputFormat.addInputPath(job, new Path(args[0]));
// FileOutputFormat.setOutputPath(job, new Path(args[1]));
Path pathInput = new Path(
  "hdfs://192.168.213.133:54310/weatherInputData/input_temp.txt");
Path pathOutputDir = new Path(
  "hdfs://192.168.213.133:54310/user/hduser1/testfs/output_mapred3");
FileInputFormat.addInputPath(job, pathInput);
FileOutputFormat.setOutputPath(job, pathOutputDir);

try {
 System.exit(job.waitForCompletion(true) ? 0 : 1);
} catch (Exception e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
}
}
```

```
}
```

INPUT: -

Set of Weather Data over the years

3. Result/Output/Writing Summary:




```
Cold Day 20151216        5.8
Cold Day 20151217        3.1
Cold Day 20151218        0.0
Cold Day 20151219        4.1
Cold Day 20151225        9.3
Cold Day 20151227        0.4
Cold Day 20151228        -0.1
Cold Day 20151229        -0.1
Cold Day 20151230        4.0
Cold Day 20151231        2.5
Hot Day 20150303         9999.0
Hot Day 20150305         9999.0
Hot Day 20150609         9999.0
Hot Day 20150613         9999.0
Hot Day 20150615         9999.0
Hot Day 20150617         9999.0
Hot Day 20150713         35.5
Hot Day 20150714         36.0
Hot Day 20150718         35.4
Hot Day 20150719         35.5
Hot Day 20150720         36.0
Hot Day 20150721         36.2
Hot Day 20150722         35.3
```

4. Learning outcomes (What I have learned):

a) Learned about working of the map reduce to extract & mines data.
b) Learned to generate map reduce program that mines weather data.