Aim:

Map Reduce Program to analyze time-temperature statistics.

1. The system receives temperatures of various cities (Austin, Boston, etc.) of USA captured at regular intervals of time on each day in an input file.

2. System will process the input data file and generates a report with Maximum and Minimum temperatures of each day along with time.

3. Generates a separate output report for each city.
Ex: Austin-r-00000
Boston-r-00000
Newjersy-r-00000
Baltimore-r-00000
California-r-00000
Newyork-r-00000

Objective:

Learn about MapReduce framework in Hadoop.
Implement MapReduce Program to analyze time-temperature statistics.

Theory:

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce.
Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.
The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

MapReduce majorly has the following three Classes. They are:

Mapper Class
The first stage in Data Processing using MapReduce is the Mapper Class. Here, RecordReader processes each Input record and generates the respective key-value pair. Hadoop's Mapper store saves this intermediate data into the local disk.

- Input Split

It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.

- Record Reader

It interacts with the Input split and converts the obtained data in the form of Key-Value Pairs.

Reducer Class
The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the HDFS.

Driver Class
The major component in a MapReduce job is a Driver Class. It is responsible for setting up a MapReduce Job to run-in Hadoop. We specify the names of Mapper and Reducer Classes long with data types and their respective job names.

Advantages of MapReduce:

1. Parallel Processing:
In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount.

2. Data Locality:
Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed the following issues:

- Moving huge data to processing is costly and deteriorates the network performance.
- Processing takes time as the data is processed by a single unit which becomes the bottleneck.
- The master node can get over-burdened and may fail.

Now, MapReduce allows us to overcome the above issues by bringing the processing unit to the data. So, as you can see in the above image that the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:

- It is very cost-effective to move processing unit to the data.

- The processing time is reduced as all the nodes are working with their part of the data in parallel.
- Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

MapReduce Program to analyze time-temperature statistics:

In **map method**, we are parsing each input line and maintain a counter for extracting date and each temperature & time information. For a given input line, first extract date (counter ==0) and followed by alternatively extract time (counter%2==1) since time is on odd number position like (1,3,5....) and get temperature otherwise. Compare for max & min temperature and store it accordingly. Once while loop terminates for a given input line, write **maxTempTime** and **minTempTime** with date.

In **reduce method**, for each reducer task, setup method is executed and create MultipleOutput object. For a given key, we have two entries (maxtempANDTime and mintempANDTime). Iterate values list, split value and get temperature & time value. Compare temperature value and create actual value sting which reducer write in appropriate file.

In **main method**, an instance of Job is created with Configuration object. Job is configured with mapper, reducer class and along with input and output format. MultipleOutputs information added to Job to indicate file name to be used with input format. For this sample program, we are using input file ("/weatherInputData/input_temp.txt") placed on HDFS and output directory (/user/hduser1/testfs/output_mapred5) will be also created on HDFS. Refer below command to copy downloaded input file from local file system to HDFS and give write permission to client who is executing this program unit so that output directory can be created.

Code of Program:

```
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;

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();
 }
}
}
```

Execution:

Copy a input file form local file system to HDFS

```
hdoop@benoi:~/hadoop-3.2.1/bin$ ./hadoop fs -put
/home/zytham/input_temp.txt /weatherInputData/
```

Give write permission to all user for creating output directory

```
hdoop@benoi:~/hadoop-3.2.1/bin$ ./hadoop fs -chmod -R 777
/user/hduser1/testfs/
```

Before executing above program unit make sure hadoop services are running(to start all service execute ./start-all.sh from <hadoop_home>/sbin).
Now execute above sample program. Run -> Run as hadoop. Wait for a moment and check whether output directory is in place on HDFS. Execute following command to verify the same.

Output:

```
hdoop@benoi:~/hadoop-3.2.1/bin$ ./hadoop fs -ls
/user/hduser1/testfs/output_mapred3 Found 8
items
-rw-r--r--   3 zytham supergroup        438 2020-12-11 19:21
/user/hduser1/testfs/output_mapred3/Austin-r-00000
-rw-r--r--   3 zytham supergroup        219 2020-12-11 19:21
/user/hduser1/testfs/output_mapred3/Baltimore-r-00000
-rw-r--r--   3 zytham supergroup        219 2020-12-11 19:21
/user/hduser1/testfs/output_mapred3/Boston-r-00000
-rw-r--r--   3 zytham supergroup        511 2020-12-11 19:21
/user/hduser1/testfs/output_mapred3/California-r-00000
-rw-r--r--   3 zytham supergroup        146 2020-12-11 19:21
/user/hduser1/testfs/output_mapred3/Newjersy-r-00000
-rw-r--r--   3 zytham supergroup        219 2020-12-11 19:21
/user/hduser1/testfs/output_mapred3/Newyork-r-00000
-rw-r--r--   3 zytham supergroup          0 2020-12-11 19:21
/user/hduser1/testfs/output_mapred3/_SUCCESS
-rw-r--r--   3 zytham supergroup          0 2020-12-11 19:21
/user/hduser1/testfs/output_mapred3/part-r-00000
```

Open one of the file and verify expected output schema, execute following command for the same.

```
hdoop@benoi:~/hadoop-3.2.1/bin$ ./hadoop fs -cat
/user/hduser1/testfs/output_mapred3/Austin-r-00000
25-Jan-2020 Time: 12:34:542 MinTemp: -22.3 Time: 05:12:345 MaxTemp:
35.7
26-Jan-2020 Time: 22:00:093 MinTemp: -27.0 Time: 05:12:345 MaxTemp:55.7
27-Jan-2020 Time: 02:34:542 MinTemp: -22.3 Time: 05:12:345 MaxTemp:55.7
29-Jan-2020 Time: 14:00:093 MinTemp: -17.0 Time: 02:34:542 MaxTemp:62.9
30-Jan-2020 Time: 22:00:093 MinTemp: -27.0 Time: 05:12:345 MaxTemp:49.2
31-Jan-2020 Time: 14:00:093 MinTemp: -17.0 Time: 03:12:187 MaxTemp:56.0
```

Conclusion:

In this assignment, we have successfully implemented MapReduce program to analyze time-temperature statistics.