Experiment No:-1

1)What is Big Data Engineering and its applications.

Big data engineering is a field that focuses on the design, development, and maintenance of systems and infrastructure to handle large volumes of data. This involves collecting, storing, processing, and analyzing data to extract valuable insights and support decision-making processes.

Some of the key aspects of big data engineering include:

Data Collection: Gathering data from various sources such as sensors, social media, web logs, and databases.

Data Storage: Storing the collected data efficiently in distributed storage systems like Hadoop Distributed File System (HDFS), Amazon S3, or Google Cloud Storage.

Data Processing: Processing the data using distributed computing frameworks like Apache Spark, Apache Flink, or Apache Hadoop MapReduce. This includes tasks like cleaning, transforming, and aggregating the data.

Data Analysis: Analyzing the processed data to derive meaningful insights using techniques such as machine learning, data mining, and statistical analysis.

Data Visualization: Presenting the analyzed data in a visually appealing and easyto-understand format using tools like Tableau, Power BI, or matplotlib in Python.

Applications of big data engineering are vast and span across various industries:

E-commerce: Recommender systems use big data to suggest products to customers based on their past behavior and preferences.

Healthcare: Analyzing large volumes of medical data can help in disease detection, drug discovery, and personalized medicine.

Finance: Detecting fraudulent transactions, managing risk, and making investment decisions based on market trends are common applications of big data in finance.

Manufacturing: Predictive maintenance uses big data to anticipate equipment failures and schedule maintenance before breakdowns occur, reducing downtime and costs.

Smart Cities: Big data can be used to optimize traffic flow, manage energy consumption, and improve public services in urban areas.

Social Media: Analyzing user interactions and content can help in targeted advertising, sentiment analysis, and trend prediction.

Telecommunications: Analyzing call detail records and network data can help in optimizing network performance, detecting anomalies, and improving customer experience.

These are just a few examples, and the applications of big data engineering continue to grow as more industries recognize the value of harnessing large volumes of data for insights and decision-making.

Certainly! Let's consider an industrial example: a manufacturing company that produces consumer electronics. In this scenario, big data engineering can play a significant role in various aspects of the manufacturing process:

Supply Chain Optimization: The company can use big data to optimize its supply chain by analyzing historical data on supplier performance, transportation times, and inventory levels. This analysis can help in predicting demand, managing inventory levels more efficiently, and reducing lead times.

Quality Control: Big data analytics can be employed to monitor the production process in realtime, collecting data from sensors installed on manufacturing equipment. By analyzing this data, the company can detect deviations from quality standards early on, identify root causes of defects, and take corrective actions to improve product quality.

Predictive Maintenance: Similar to the example mentioned earlier, the company can implement predictive maintenance strategies to minimize downtime and reduce maintenance costs. By analyzing data from sensors embedded in machinery, such as temperature sensors, vibration sensors, and pressure sensors, the company can predict when equipment is likely to fail and schedule maintenance proactively.

Product Design and Innovation: Big data analytics can be used to gather feedback from customers through various channels such as social media, product reviews, and customer service interactions. By analyzing this unstructured data, the company can identify emerging trends, customer preferences, and areas for product improvement, guiding future product development efforts.

Energy Efficiency: By collecting data on energy consumption across the manufacturing facility, the company can identify opportunities to optimize energy usage and reduce costs. For example, analyzing data from smart meters and energy management systems can help in identifying energyintensive processes and implementing energy-saving measures.

Demand Forecasting and Inventory Management: Big data analytics can be used to analyze sales data, market trends, and external factors such as economic indicators and weather patterns to forecast demand more accurately. This enables the company to adjust production levels and inventory levels accordingly, minimizing stockouts and excess inventory.

Overall, big data engineering can revolutionize the manufacturing industry by enabling datadriven decision-making, improving operational efficiency, and driving innovation.

Experiment No:-2

2)Perform the Installation of Hadoop

Environment required for Hadoop: The production environment of Hadoop is UNIX, but it can also be used in Windows using Cygwin. Java 1.6 or above is needed to run Map Reduce Programs. For Hadoop installation from tar ball on the UNIX environment you need

1. Java Installation
2. SSH installation
3. Hadoop Installation and File Configuration

1) Java Installation

Step 1. Type "java -version" in prompt to find if the java is installed or not. If not then download java from http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html . The tar filejdk-7u71-linux-x64.tar.gz will be downloaded to your system.

Step 2. Extract the file using the below command

1. #tar zxf jdk-7u71-linux-x64.tar.gz

Step 3. To make java available for all the users of UNIX move the file to /usr/local and set the path. In the prompt switch to root user and then type the command below to move the jdk to /usr/lib.

1. # mv jdk1.7.0_71 /usr/lib/

Now in ~/.bashrc file add the following commands to set up the path.

1. # export JAVA_HOME=/usr/lib/jdk1.7.0_71
2. # export PATH=PATH:$JAVA_HOME/bin

Now, you can check the installation by typing "java -version" in the prompt.

2) SSH Installation

SSH is used to interact with the master and slaves computer without any prompt for password. First of all create a Hadoop user on the master and slave systems

1. # useradd hadoop
2. # passwd Hadoop

To map the nodes open the hosts file present in /etc/ folder on all the machines and put the ip address along with their host name.

1. # vi /etc/hosts

Enter the lines below

1. 190.12.1.114   hadoop-master
2. 190.12.1.121   hadoop-salve-one
3. 190.12.1.143   hadoop-slave-two

Set up SSH key in every node so that they can communicate among themselves without password. Commands for the same are:

1. # su hadoop
2. $ ssh-keygen -t rsa
3. $ ssh-copy-id -i ~/.ssh/id_rsa.pub tutorialspoint@hadoop-master
4. $ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp1@hadoop-slave-1
5. $ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp2@hadoop-slave-2
6. $ chmod 0600 ~/.ssh/authorized_keys
7. $ exit

3) Hadoop Installation

Hadoop can be downloaded from http://developer.yahoo.com/hadoop/tutorial/module3.html

Now extract the Hadoop and copy it to a location.

1. $ mkdir /usr/hadoop
2. $ sudo tar vxzf hadoop-2.2.0.tar.gz ?c /usr/hadoop

Change the ownership of Hadoop folder

1. $sudo chown -R hadoop usr/hadoop

Change the Hadoop configuration files:

All the files are present in /usr/local/Hadoop/etc/hadoop

1) In hadoop-env.sh file add

1. export JAVA_HOME=/usr/lib/jvm/jdk/jdk1.7.0_71

2) In core-site.xml add following between configuration tabs,

1. **<configuration>**

2. **<property>**
3. **<name>**fs.default.name**</name>**
4. **<value>**hdfs://hadoop-master:9000**</value>**
5. **</property>**
6. **<property>**
7. **<name>**dfs.permissions**</name>**
8. **<value>**false**</value>**
9. **</property>**
10. **</configuration>**

3) In hdfs-site.xmladd following between configuration tabs,

1. **<configuration>**
2. **<property>**
3. **<name>**dfs.data.dir**</name>**
4. **<value>**usr/hadoop/dfs/name/data**</value>**
5. **<final>**true**</final>**
6. **</property>**
7. **<property>**
8. **<name>**dfs.name.dir**</name>**
9. **<value>**usr/hadoop/dfs/name**</value>**
10. **<final>**true**</final>**
11. **</property>**
12. **<property>**
13. **<name>**dfs.replication**</name>**
14. **<value>**1**</value>**
15. **</property>**
16. **</configuration>**

4) Open the Mapred-site.xml and make the change as shown below

1. **<configuration>**
2. **<property>**
3. **<name>**mapred.job.tracker**</name>**
4. **<value>**hadoop-master:9001**</value>**
5. **</property>**
6. **</configuration>**

5) Finally, update your $HOME/.bahsrc

1. cd $HOME
2. vi .bashrc
3. Append following lines in the end and save and exit
4. #Hadoop variables
5. export JAVA_HOME=/usr/lib/jvm/jdk/jdk1.7.0_71
6. export HADOOP_INSTALL=/usr/hadoop
7. export PATH=$PATH:$HADOOP_INSTALL/bin
8. export PATH=$PATH:$HADOOP_INSTALL/sbin
9. export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
10. export HADOOP_COMMON_HOME=$HADOOP_INSTALL
11. export HADOOP_HDFS_HOME=$HADOOP_INSTALL
12. export YARN_HOME=$HADOOP_INSTALL

On the slave machine install Hadoop using the command below

1. # su hadoop
2. $ cd /opt/hadoop
3. $ scp -r hadoop hadoop-slave-one:/usr/hadoop 4. $ scp -r hadoop hadoop-slave-two:/usr/Hadoop

Configure master node and slave node

1. $ vi etc/hadoop/masters
2. hadoop-master
3. $ vi etc/hadoop/slaves
4. hadoop-slave-one
5. hadoop-slave-two

After this format the name node and start all the deamons

1. # su hadoop
2. $ cd /usr/hadoop
3. $ bin/hadoop namenode -format
4. $ cd $HADOOP_HOME/sbin
5. $ start-all.sh

Experiment No:-3

3)Perform Different file management task in Hadoop

1)Create a directory in HDFS at given path(s).

Usage: hadoop fs -mkdir <paths>

Example : hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2

2)List the contents of a directory.

Usage : hadoop fs -ls <args>

Example: hadoop fs -ls /user/saurzcode

3)Upload and download a file in HDFS.
Upload:
hadoop fs -put:
Copy single src file, or multiple src files from local file system to the Hadoop data file system

Usage: hadoop fs -put <localsrc> ...
<HDFS_dest_Path>

Example: hadoop fs -put /home/saurzcode/Samplefile.txt /user/ saurzcode/dir3/
Download

hadoop fs -get:
Copies/Downloads files to the local file system

Usage: hadoop fs -get <hdfs_src>
<localdst>

Example: hadoop fs -get
/user/saurzcode/dir3/Samplefile.txt /home/

4)See contents of a file Same
as unix cat command:
Usage:
hadoop fs -cat <path[filename]>
Example: hadoop fs -cat
/user/saurzcode/dir1/abc.txt

5)Copy a file from source to destination

This command allows multiple sources as well in which case the destination must be a directory.

Usage:
hadoop fs -cp <source> <dest> Example: hadoop fs -cp /user/saurzcode/dir1/abc.txt /user/saurzcode/ dir2

6)Copy a file from/To Local file system to HDFS copyFromLocal

Usage: hadoop fs -copyFromLocal <localsrc> URI

Example: hadoop fs -copyFromLocal /home/saurzcode/abc.txt /user/ saurzcode/abc.txt

Similar to put command, except that the source is restricted to a local file reference.

copyToLocal

Usage: hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

Similar to get command, except that the destination is restricted to a local file reference.

7)Move file from source to destination. Note:-
Moving files across filesystem is not permitted.

Usage : hadoop fs -mv <src> <dest>

Example: hadoop fs -mv /user/saurzcode/dir1/abc.txt /user/saurzcode/ dir2

8)Remove a file or directory in HDFS.
Remove files specified as argument. Deletes directory only when it is empty

Usage : hadoop fs -rm <arg>

Example: hadoop fs -rm /user/saurzcode/dir1/abc.txt

9)Recursive version of delete.

Usage :

hadoop fs -rmr <arg> Example:
hadoop fs -rmr
/user/saurzcode/ **10)Display last few lines of a file.** Similar to tail command in Unix.

Usage :

hadoop fs -tail <path[filename]>

Example: hadoop fs -tail
/user/saurzcode/dir1/abc.txt

11)Display the aggregate length of a file.

Usage : hadoop fs -
du <path>

Example: hadoop fs -du
/user/saurzcode/dir1/abc.txt

Experiment No:-4

4.Word Count Map Reduce program to understand Map Reduce Paradigm

Source code:

```
import java.io.IOException; import java.util.StringTokenizer; 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
org.apache.hadoop.mapreduce.Reducer; import
org.apache.hadoop.conf.Configuration; import
org.apache.hadoop.mapreduce.Job; import
```

```
org.apache.hadoop.mapreduce.lib.input.TextInputFormat; import
org.apache.hadoop.mapreduce.lib.output.TextOutputFormat; import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; import
org.apache.hadoop.fs.Path;  public class WordCount
{ public static class Map extends Mapper<LongWritable,Text,Text,IntWritable> { public void
map(LongWritable key, Text value,Context context) throws
IOException,InterruptedException{
String line = value.toString(); StringTokenizer tokenizer = new
StringTokenizer(line); while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken()); context.write(value, new IntWritable(1));
}
} } 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 x: values)
{
sum+=x.get();
} context.write(key, new IntWritable(sum));
}
}

public static void main(String[] args) throws Exception {
Configuration conf= new Configuration(); Job job = new
Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class); 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 to 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);
}
}

The entire MapReduce program can be fundamentally divided into three parts:

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

We will understand the code for each of these three parts sequentially.

Mapper code: public static

```
class Map extends

Mapper<LongWritable,Text,Text,IntWritable> {

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

String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line); while
(tokenizer.hasMoreTokens()) { value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
```

- We have created a class Map that extends the class Mapper which is already defined in the MapReduce Framework.

- We define the data types of input and output key/value pair after the class declaration using angle brackets.
- Both the input and output of the Mapper is a key/value pair.
- Input:
  ◦ The key is nothing but the offset of each line in the text file:LongWritable
  ◦ The value is each individual line (as shown in the figure at the right): Text
- Output:
  ◦ The key is the tokenized words: Text
  ◦ We have the hardcoded value in our case which is 1: IntWritable ◦
  Example – Dear 1, Bear 1, etc.
- We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to 1.

<u>Reducer Code:</u>

```
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 x:
values)
{ sum+=x.get();
} context.write(key, new
IntWritable(sum));}}
```

- We have created a class Reduce which extends class Reducer like that of Mapper.
- We define the data types of input and output key/value pair after the class declaration using angle brackets as done for Mapper.
- Both the input and the output of the Reducer is a keyvalue pair.
- Input:

◦ The key nothing but those unique words which have been generated after the sorting and shuffling phase: Text

◦ The value is a list of integers corresponding to each key: IntWritable ◦ Example – Bear, [1, 1], etc.

- Output:

  ◦ The key is all the unique words present in the input text file: Text

  ◦ The value is the number of occurrences of each of the unique words: IntWritable ◦ Example – Bear, 2; Car, 3, etc.

- We have aggregated the values present in each of the list corresponding to each key and produced the final answer.

- In general, a single reducer is created for each of the unique words, but, you can specify the number of reducer in mapred-site.xml.

Driver Code:

```
Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program"); job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class); job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class); job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
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]));
```

- In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
- We specify the name of the job , the data type of input/ output of the mapper and reducer.

- We also specify the names of the mapper and reducer classes.
- The path of the input and output folder is also specified.
- The method setInputFormatClass () is used for specifying that how a Mapper will read the input data or what will be the unit of work. Here, we have chosen TextInputFormat so that single line is read by the mapper at a time from the input text file.
- The main () method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.

Run the MapReduce code:

The command for running a MapReduce code is: hadoop jar hadoop-mapreduce-

example.jar WordCount / sample/input /sample/output

Experiment No:-5

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

Problem Statement:

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

Expected output:-

In each output file record should be like this:
25-Jan-2014

Time: 12:34:54 MinTemp: -22.3 Time:
05:12:34 MaxTemp: 35.7

First download input file which contains temperature statistics with time for multiple cities.Schema of record set : CA_25-Jan-2014 00:12:345 15.7 01:19:345 23.1 02:34:542 12.3 ...... CA is city code, here it stands for California followed by date. After that each pair of values represent time and temperature.

Mapper class and map method:-

The very first thing which is required for any map reduce problem is to understand what will be the type of keyIn, ValueIn, KeyOut,ValueOut for the given Mapper class and followed by type of map method parameters.

- public class WhetherForcastMapper extends Mapper <**Object, Text, Text, Text**>
- Object (keyIn) - Offset for each line, line number 1, 2...
- Text **(**ValueIn**) -** Whole string for each line (CA_25-Jan-2014 00:12:345 ......)
- Text **(**KeyOut**) -** City information with date information as string
- Text **(**ValueOut**) -** Temperature and time information which need to be passed to reducer as string.
- public void map(**Object keyOffset, Text dayReport, Context con**) { } • *KeyOffset* is like line number for each line in input file.
- *dayreport* is input to map method - whole string present in one line of input file.
- *con* is context where we write mapper output and it is used by reducer.

Reducer class and reducer method:-

Similarly,we have to decide what will be the type of keyIn, ValueIn,

KeyOut,ValueOut for the given Reducer class and followed by type of reducer method parameters.

- public class WhetherForcastReducer extends Reducer<**Text, Text, Text, Text**>
- Text(keyIn) - it is same as keyOut of Mapper.
- Text(ValueIn)- it is same as valueOut of Mapper.
- Text(KeyOut)- date as string
- text(ValueOut) - reducer writes max and min temperature with time as string
- public void reduce(**Text key, Iterable<Text> values, Context context**)
- Text key is value of mapper output. i.e:- City & date information
- Iterable<Text> values - values stores multiple temperature values for a given city and date.
- context object is where reducer write it's processed outcome and finally written in file.

MultipleOutputs :- In general, reducer generates output file(i.e: part_r_0000), however in this use case we want to generate multiple output files. In order to deal with such scenario we need to use MultipleOutputs of "org.apache.hadoop.mapreduce.lib.output.MultipleOutputs" which provides a way to write multiple file depending on reducer outcome. See below reducer class for more details.For each reducer task multipleoutput object is created and key/result is written to appropriate file.

Lets create a Map/Reduce project in eclipse and create a class file name it as CalculateMaxAndMinTemeratureWithTime. For simplicity,here we have written mapper and reducer class as inner static class. Copy following code lines and paste in newly created class file.

* Question:- To find Max and Min temperature from record set stored in
* text file. Schema of record set :- tab separated (\t) CA_25-Jan-2014
* 00:12:345 15.7 01:19:345 23.1 02:34:542 12.3 03:12:187 16 04:00:093 14 05:12:345 35.7 06:19:345 23.1 07:34:542 12.3 08:12:187 16
* 09:00:093 -7 10:12:345 15.7 11:19:345 23.1 12:34:542 -22.3 13:12:187 16 14:00:093 -7 15:12:345 15.7 16:19:345 23.1 19:34:542 12.3
* 20:12:187 16 22:00:093 -7
* Expected output:-
* Creates files for each city and store maximum & minimum
* temperature for each day along with time.

```
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.MultipleOutput s; 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.FileOutputForm at; import
org.apache.hadoop.mapreduce.lib.output.TextOutputForm at;

/**
* @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 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;
 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.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, OutputName)

  }
```

```
 }
}
```

```
// 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();
  }
}
}
```

Explanation

In **map method**, we are parsing each input line and maintains 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 entry (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**,a 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.

Copy a input file form local file system to HDFS

hduser1@ubuntu:/usr/local/hadoop2.6.1/bin$ ./ adoop fs -put /home/zytham/input_temp.txt / weatherInputData/

Give write permission to all user for creating output directory

hduser1@ubuntu:/usr/local/hadoop2.6.1/bin$ ./ adoop fs -chmod -R 777 /user/hduser1/testfs/

Before executing above program unit make sure adoop services are running(to start all service execute ./start-all.sh from < adoop_home>/sbin).

Now execute above sample program. Run -> Run as adoop. Wait for a moment and check whether output directory is in place on HDFS. Execute following command to verify the same.

hduser1@ubuntu:/usr/local/hadoop2.6.1/bin$ ./adoop fs -ls /user/hduser1/testfs/output_mapred3

Found 8 items

-rw-r—r-- 3 zytham supergroup 438
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
Austin-r-00000

-rw-r—r-- 3 zytham supergroup 219
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
Baltimore-r-00000

-rw-r—r-- 3 zytham supergroup 219
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
Boston-r-00000

-rw-r—r-- 3 zytham supergroup 511
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
California-r-00000

-rw-r—r-- 3 zytham supergroup 146
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
Newjersy-r-00000

-rw-r—r-- 3 zytham supergroup 219
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
Newyork-r-00000

-rw-r—r-- 3 zytham supergroup 0
2015-12-11 19:21 /user/hduser1/testfs/output_mapred3/
_SUCCESS

-rw-r—r-- 3 zytham supergroup 0
2015-12-11 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.

hduser1@ubuntu:/usr/local/hadoop2.6.1/bin$ ./adoop fs -cat /user/hduser1/testfs/output_mapred3/
Austin-r-00000

25-Jan-2014 Time: 12:34:542 MinTemp: -22.3 Time: 05:12:345 MaxTemp: 35.7

Note:-

- In order to reference input file from local file system instead of HDFS, uncomment below lines in main method and comment below added addInputPath and setOutputPath lines. Here Path(args[0]) and Path(args[1]) read input and output location path from program arguments. OR create path object with sting input of input file and output location.
  // FileInputFormat.addInputPath(job, new Path(args[0]));
  // FileOutputFormat.setOutputPath(job, new Path(args[1]));

Execute WeatherReportPOC.jar on single node cluster

We can create jar file out of this project and run on single node cluster too. Download WeatherReportPOC jar and place at some convenient location.Start hadoop services(./start-all.sh from <hadoop_home>/sbin). I have placed jar at "/home/zytham/Downloads/WeatherReportPOC.jar". Execute following command to submit job with input file HDFS location is "/ wheatherInputData/input_temp.txt" and output directory location is "/user/ hduser1/testfs/output_mapred7" **hduser1@ubuntu:/usr/local/hadoop2.6.1/bin$**

./hadoop

jar /home/zytham/Downloads/

WeatherReportPOC.jar

CalculateMaxAndMinTemeratureWithTime / wheatherInputData/input_temp.txt

/user/hduser1/ testfs/output_mapred7

15/12/11 22:16:12 INFO Configuration.deprecation: session.id
is deprecated. Instead, use dfs.metrics.session-id

15/12/11 22:16:12 INFO jvm.JvmMetrics: Initializing
JVM Metrics with processName=JobTracker, sessionId= 15/12/11 22:16:14 WARN
mapreduce.JobResourceUploader:

Hadoop command-line option parsing not performed.

Implement the Tool interface and execute your application with ToolRunner to remedy this.

....

15/12/11 22:16:26 INFO output.FileOutputCommitter:

Saved output of task
'attempt_local1563851561_0001_r_000000_0' to hdfs://
hostname:54310/user/hduser1/testfs/output_mapred7/
_temporary/0/task_local1563851561_0001_r_000000

15/12/11 22:16:26 INFO mapred.LocalJobRunner: reduce

> reduce

15/12/11 22:16:26 INFO mapred.Task: Task 'attempt_local1563851561_0001_r_000000_0' done.

15/12/11 22:16:26 INFO mapred.LocalJobRunner:
Finishing task: attempt_local1563851561_0001_r_000000_0 15/12/11 22:16:26 INFO
mapred.LocalJobRunner: reduce task executor complete.

15/12/11 22:16:26 INFO mapreduce.Job: map 100% reduce 100%

15/12/11 22:16:27 INFO mapreduce.Job: Job job_local1563851561_0001 completed successfully
15/12/11 22:16:27 INFO mapreduce.Job: Counters: 38 ……

Experiment No:-6

6)Implementing Matrix Multiplication with Hadoop Map Reduce

```
import java.io.IOException; import java.util.*;
import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;

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 TwoStepMatrixMultiplication {

 public static class Map extends Mapper<LongWritable, Text, Text, Text>
{        public void map(LongWritable key, Text value, Context  context)
throws IOException, InterruptedException {
          String line = value.toString();
          String[] indicesAndValue = line.split(",");
          Text outputKey = new Text();
          Text outputValue = new Text();            if
(indicesAndValue[0].equals("A")) {
outputKey.set(indicesAndValue[2]);
             outputValue.set("A," + indicesAndValue[1] + ","
+  indicesAndValue[3]);               context.write(outputKey,
outputValue);
          } else {
outputKey.set(indicesAndValue[1]);
             outputValue.set("B," + indicesAndValue[2] + ","
+  indicesAndValue[3]);               context.write(outputKey,
outputValue);
          }
       }
}

   public static class Reduce extends Reducer<Text, Text, Text, Text> {
      public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
          String[] value;
 ArrayList<Entry<Integer, Float>> listA = new
```

```
ArrayList<Entry<Integer, Float>>();
          ArrayList<Entry<Integer, Float>> listB = new
ArrayList<Entry<Integer, Float>>();
 for (Text val : values) {                 value =
val.toString().split(",");                 if
(value[0].equals("A")) {
listA.add(new SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));
             } else {
                listB.add(new SimpleEntry<Integer,
Float>(Integer.parseInt(value[1]), Float.parseFloat(value[2])));
             }
          }            String i;
float a_ij;             String k;
float b_jk;
          Text outputValue = new Text();            for (Entry<Integer, Float> a : listA)
{               i = Integer.toString(a.getKey());
             a_ij = a.getValue();
             for (Entry<Integer, Float> b : listB) {                  k =
Integer.toString(b.getKey());
                b_jk = b.getValue();
                outputValue.set(i + "," + k + ","
+  Float.toString(a_ij*b_jk));
context.write(null, outputValue);
             }
          }
       }
}

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

      Job job = new Job(conf,  "MatrixMatrixMultiplicationTwoSteps");
      job.setJarByClass(TwoStepMatrixMultiplication.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("hdfs://
127.0.0.1:9000/matrixin"));
             FileOutputFormat.setOutputPath(job, new Path("hdfs://
127.0.0.1:9000/matrixout"));
```

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