Assignment 4: Weather

* Aim: Design & develop a distributed application to find the coolest/hottest year from available weather data. Process it using MapReduce.

* Theory:

Q1) List & explain commands used for execution of any mapreduce program.

Ans: i) Download the input file/dataset and store this on HDFS

Syntax: hdfs dfs -copyFromLocal /path/to/input /pathHDFS
Example: hdfs dfs -copyFromLocal /Downloads/input.txt /PVG/input.txt

Alternatively, we can use the -put command to transfer file from local system to HDFS.

ii) Create output directory in HDFS

Syntax: hdfs dfs -mkdir /path
Example: hdfs dfs -mkdir /PVG/Output

iii) Create source code. Write the code for Mapper, Reducer & Driver classes.

iv) Create JAR file.

For IntelliJ IDE, go to File → Module Settings → Artifacts → New → From module with dependencies → JAR file → select Driver.java → Apply changes → OK.

Then, go to Build → Build with artifacts.

After this step, jar file will be created in Downloads folder.

v) Execute jar file using hadoop jar command.

Syntax: hadoop jar /path/to/jar/file Driver class /path/to/input/file /path/to/output/directory

Example: ~~hd~~ hadoop jar ~~Words.ja~~/home/hduser/Words.jar Words.Driver.class /PVG/input.txt /PVG/Output1

vi) Display output using cat command

Syntax: hdfs dfs -cat /path/to/file

Example: hdfs dfs -cat /PVG/Output1/part-*

Q2) Explain following classes:

1) Job

Ans: i) The Hadoop job class represents the unit of work that is sent to the mapreduce programming model.

ii) It allows user to configure the job, control its execution and query the state. User creates application, describes the job & then submits job & monitors its progress

iii)
```
Job job = Job.getInstance();
Job job = new Job(new Configuration(), "Name");
```

2) FileInputFormat

Ans: i) FileInputFormat is the base class for all File based Input formats. This provides a generic implementation of getSplits().

ii) FileInputFormat is used to provide input file to Hadoop job.

iii) FileInputFormat.addInputPath(job, new Path("..."));

3) FileOutputFormat

Ans: i) FileOutputFormat is the base class for OutputFormats that read

from FileSystems.

ii) FileOutputFormat.addOutputPath(job, new Path(".."));

4) FileStatus

Ans: i) FileStatus is an interface that represents the client side information for a file.

ii) Output of fs.listStatus(path) returns an array of Filestatus objects in which arr[0] is SUCCESS-FILE and arr[1] contains file information.

iii) This information includes name, size, path, block size, permissions, etc.

5) LongWritable

Ans: i) LongWritable class in Hadoop wraps the Java Long class which is used for storing large numbers.

ii) LongWritable is a serializable class and is similar to IntWritable in terms of implementation.

* Conclusion: Thus we have successfully implemented MapReduce for Weather Data.

Mapper class : WeatherMapper.java

```
package Weather;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class  WeatherMapper extends Mapper<LongWritable,Text,Text,IntWritable>
{
    public void map(LongWritable key,Text value,Context context)  throws IOExc
eption,InterruptedException {
        String line = value.toString();
        String year = line.substring(15, 19);
        int temp = 9999;

        if (line.charAt(87)=='+') {
            temp= Integer.parseInt(line.substring(88, 92));
        } else {
            temp= Integer.parseInt(line.substring(87, 92));
        }

        if (temp != 9999) {
            context.write(new Text(year), new IntWritable(temp));
        }
    }
}
```

Reducer class : WeatherReducer.java

```
package Weather;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
```

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

import java.io.IOException;

public class WeatherReducer extends Reducer<Text, IntWritable, Text, IntWritab
le> {

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

        int max = -9999;
        int min = 9999;

        for(IntWritable value : values) {
            if(value.get() < min)
                min = value.get();
            if(value.get() > max)
                max = value.get();
        }
        context.write(key, new IntWritable(max));
        context.write(key, new IntWritable(min));
    }

}
```

Driver class : WeatherDriver.java

```
package Weather;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
public class WeatherDriver {

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

        Configuration con = new Configuration();
        Job job = new Job(con, "Weather");

        String string;

        job.setJarByClass(Weather.WeatherDriver.class);
        job.setMapperClass(Weather.WeatherMapper.class);
        job.setReducerClass(Weather.WeatherReducer.class);

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

        FileInputFormat.addInputPath(job, new Path(args[1]));
        FileOutputFormat.setOutputPath(job, new Path(args[2]));

        job.waitForCompletion(true);

        FileSystem fs = FileSystem.get(con);
        FileStatus[] status = fs.listStatus(new Path("hdfs://localhost:9000"+a
rgs[2]));
        FSDataInputStream fd = fs.open(status[1].getPath());

        string = fd.readLine();

        float max = Integer.MIN_VALUE, min = Integer.MAX_VALUE, temp;
        String minYear = null, maxYear = null;

        while(string != null) {
            String [] tokens = string.split("\t");
            temp = Integer.parseInt(tokens[1]);

            if(temp > max) {
                max = temp;
                maxYear = tokens[0];
                continue;
```

```
            }
            if(temp < min) {
                min = temp;
                minYear = tokens[0];
            }
            string = fd.readLine();
        }

        System.out.println("Maximum temperature : " + max/10 + " in the year "
 + maxYear);
        System.out.println("Minimum temperature : " + min/10 + " in the year "
 + minYear);
    }
}
```

Output Screenshot