Omkar Gurav
TE IT Batch T3
Roll No:- 8048

Assignment - 4

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

- Theory:-

Q.1. List & explain commands used for execution of any mapreduce program.

→ ① 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.

② Create output directory in HDFS

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

③ Create source code. Write the code for Mapper, Reducer & Driver classes.

④ Create JAR file.

For Eclipse IDE, go to Java project (package) you are working on → Export → JAR file → select Java project → Finish select export destination → Finish

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

⑤ 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: hadoop jar /home/hduser/words.jar Words.Driver /PVG/input.txt /PVG/Output

⑥ Display output using cat command

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

Example: hdfs dfs -cat /PVG/Output2/part-*

Q.2 Explain following classes.

① Job

→ ① The Hadoop job class represents the unit of work that is sent to the mapreduce programming model.

② 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.

③ Job job = Job.getInstance();
Job job = new Job(new Configuration(), "Name");

② File InputFormat

→ ① FileInputFormat is the base class for all File based Input formats. This provides a generic implementations of getSplits().

② FileInputFormat is used to provide input file to Hadoop job.

③ FileInputFormat.addInputPath (job, new Path ("..."));

③ FileOutputFormat

→ ① FileOutputFormat is the base class for OutputFormats that read from FileSystems.

② FileOutputFormat.addOutputPath(job, new Path("..."));

④ FileStatus

→ ① FileStatus is an interface that represents the client side information for a file.

② Output of fs.listStatus(path) returns an array of FileStatus objects in which arr[0] is success_file & arr[2] contains file information.

③ This information includes name, size, path, block size, permissions etc.

⑤ LongWritable

→ ① LongWritable class in Hadoop wraps the Java Long class which is used for storing large numbers.

② LongWritable is a serializable class & is similar to IntWritable in terms of implementation.

- Conclusion:- Thus we have successfully implementated MapReduce for Weather Data.

Driver class : Driver.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.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver {

              @SuppressWarnings("deprecation")
           public static void main(String[] args) throws Exception{
           //creating object of configuration class
           Configuration c = new Configuration();

           //Assigning job to new configuration object
           Job job = new Job(c);

           //setting jar class
           job.setJarByClass(weather.Driver.class);

           job.setMapperClass(weather.TempMapper.class);

           job.setReducerClass(weather.TempReducer.class);

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

           //Adding a Path to the list of inputs
           FileInputFormat.addInputPath(job, new Path(args[0]));

           //Setting the Path of the output directory
           FileOutputFormat.setOutputPath(job,new Path(args[1]));

           //wait till job is completed
           job.waitForCompletion(true);

              //file system object
              FileSystem fs = FileSystem.get(c);

              FileStatus[] status = fs.listStatus(new
Path("hdfs://localhost:9000"+args[1]));
              FSDataInputStream fd = fs.open(status[1].getPath());

              String str = fd.readLine();

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

        int pos = 0;

        while(str != null) {

            String [] parts = str.split("\t");
            temp = Integer.parseInt(parts[1]);

            if(pos % 2 == 0)
            {
              if(temp > max)
              {
                max = temp;
                maxYear = parts[0];
              }
            }
            else {
              if(temp < min)
              {
                min = temp;
                minYear = parts[0];
```

```
              }
            }

            pos++;

            str = fd.readLine();
        }

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

       }
}
```

Mapper class : TempMapper.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  TempMapper extends Mapper<LongWritable,Text,Text,IntWritable>
{
    public void map(LongWritable key,Text value,Context context)  throws
IOException,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 : TempReducer.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 TempReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    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));
    }

}
```

Output Screenshots