**ASSIGNMENT 12**

**Aim:** Write a program that interacts with the weather database. Find the day and the station with the maximum snowfall in 2013.

**Objectives:** To learn the configuration and implementation of Hadoop.

### Theory:

**MapReduce:**

It divides a task into small parts and assigns them to many computers later, the results are collected at one place and integrated to form the result dataset.

MapReduce algorithm contains two important tasks, namely map and reduce.

The map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs). The reduce task takes the output from map as an input and combines those data tuples into a smaller set of tuples.The reduce task is always performed after the map job.

**Input phase:** Here we have a record reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.

**Map:** It is a user-defined function, which takes a series of key value pairs and processes each one of them to generate zero or more key-value pairs.

**Intermediate keys:** The key-value pairs generated by mapper are known as intermediate keys.

**Combiner:** A combiner is a type of local reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code. It aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is option.

**Shuffle and Sort:** The reducer task starts with the shuffle and sort step. It downloads the grouped key-value pairs onto local machine, where the reducer is running. The individual key-value pairs are sorted by the into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in reducer task.

**Reducer:** The reducer takes the grouped key-value paired data as input and mins a reducer function on each one of them. Here the data can be aggregate , filteration and combined in a number of ways , and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.

**Output phase:** In this phase, we have an output formatter that translates the final key value pairs from the Reducer function & writes them onto a file using a record writer.



**Code:**

import java.io.IOException;

import java.util.\*;

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 Snow {

public static class Map extends Mapper<LongWritable, Text, Text, FloatWritable> {

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

String line = value.toString();

String dateStation = null;

float snow = 0.0f;

StringTokenizer s = new StringTokenizer(line,",");

String date = s.nextToken();

if(date.startsWith("2013")) {

String station = s.nextToken();

String temp = s.nextToken();

temp = s.nextToken();

temp = s.nextToken();

temp = s.nextToken();

snow = Float.parseFloat(temp);

dateStation = "Date: " + date + ", Station: " + station;

if(snow!=0.0f)

context.write(new Text(dateStation), new FloatWritable(snow));

}

}

}

public static class Reduce extends Reducer<Text, FloatWritable, Text, FloatWritable> {

float max = 0.0f;

Text date = new Text();

public void reduce(Text key, Iterable<FloatWritable> values, Context context)

throws IOException, InterruptedException {

for (FloatWritable val : values) {

float num = val.get();

if(num > max) {

max = num;

date = key;

}

}

}

public void cleanup(Context context) throws IOException, InterruptedException {

context.write(new Text("Max snowfall in 2013:"),new FloatWritable(0.0f));

context.write(date, new FloatWritable(max));

}

}

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

Configuration conf = new Configuration();

Job job = new Job(conf, "Snowfall");

job.setJarByClass(Snow.class);

job.setMapperClass(Map.class);

job.setCombinerClass(Reduce.class);

job.setReducerClass(Reduce.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(FloatWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

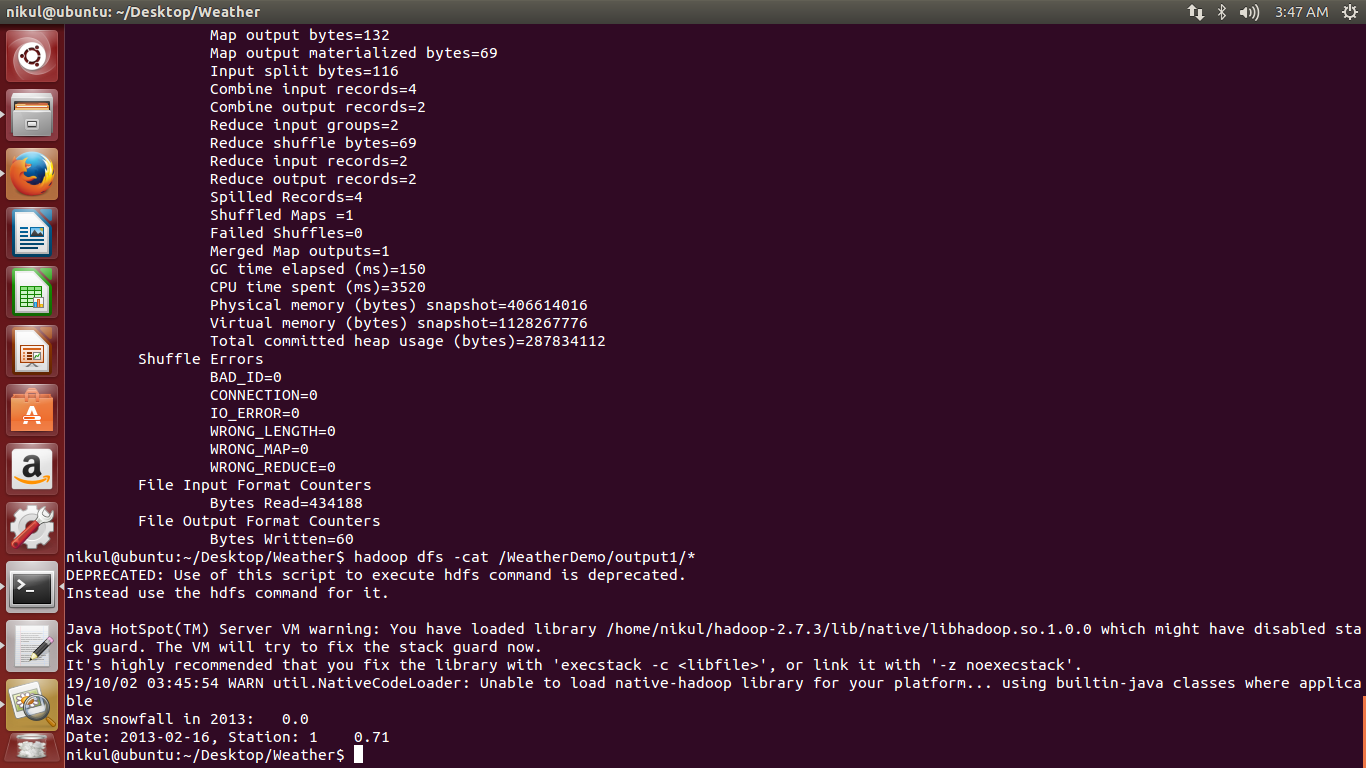
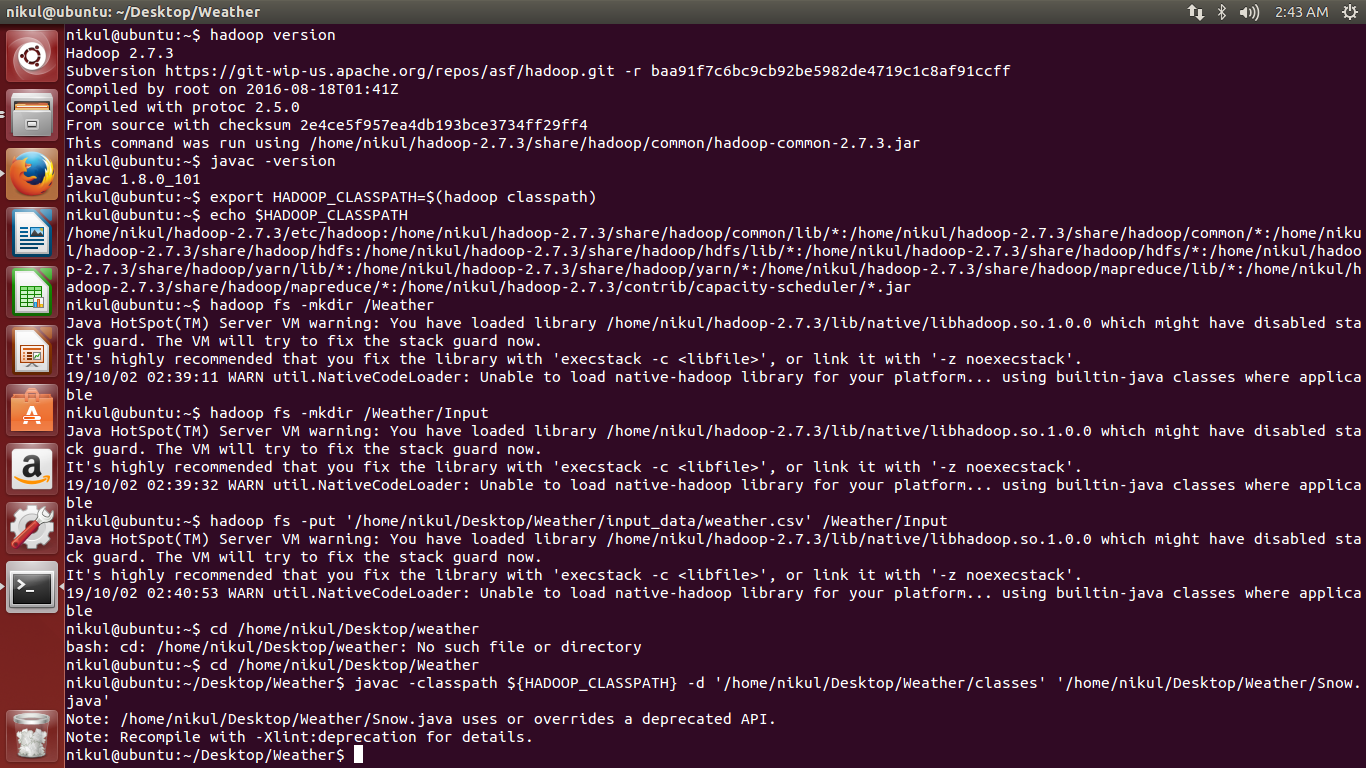
FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Output:**

****

**Conclusion:** Thus we have implemented the Hadoop mapreduce program to find the day and station having maximum snowfall.