HADOOP MapReduce Job for Finding Maximum Temperature

1. Problem Statement: Use the data from National Climate Data Center (<http://www.ncdc.noaa.gov/>) and mine that data to find maximum temperature recorded for each year.
2. Problem Description: Weather sensors of NCDC collect data every hour at many locations across the globe and gather large volume of log data. This is stored in line oriented ASCII format. Each line contains important information such as date and time of observation, latitude and longitude, temperature, quality code etc. These files are gzipped and stored on a ftp server. They are referred to as datasets.
3. Using MapReduce Framework to analyse the above data:
   1. MapReduce is a programming model for data processing.
   2. MapReduce works by breaking the process into 2 phases: map phase and reduce phase.
   3. Each phase has a key-value pairs as input and output
   4. The programmer specifies 2 functions: map and reduce
   5. The input to map phase is the raw NCDC data as txt file
   6. In the map function, we pull out the year and the air temperature. We also filter out or drop bad records with missing data
   7. The extracted data is emitted as output in the form (year, temperature)
   8. The output is processed by MapReduce framework – It sorts and groups key-value pairs by key. So the input to Reduce function is (year, [t1, t2, t3….])
   9. The reduce function just iterates through the list and picks up maximum reading
   10. Output of the reduce function: (year, maxTemp)
4. Algorithm
   1. Main Program
      1. Read the first command line argument as input text file
      2. Read the second command line argument as the output path
      3. Specify the mapper and reducer classes to be used
      4. Submit the job and wait for it to finish
   2. Mapper Program
      1. Input:
         1. input key – Offset of the beginning of the line from the beginning of the file
         2. Value – One line at a time from the dataset
      2. Extract year and temperature from Value using substring method
      3. If temperature value is missing or quality code not matching, ignore
      4. Write the year and temperature into Context. Now year becomes the key and temperature is the value
      5. Output:
         1. Text: year
         2. IntWritable: temperature
   3. Reducer Program
      1. Input
         1. Text : year
         2. IntWritable: temperature (iterable values)
      2. Iterate through each value for the given key and find maximum value using Math.max function
      3. Write the key and maximum value into the Context
5. Coding: Create 3 files: MaxTemperature.java, MaxTemperatureMapper.java and MaxTemperatureReducer.java for above programs respectively.
6. Compilation: compile all the 3 java files together using hadoopclasspath:
   1. javac –classpath `hadoopclasspath` \*.java
   2. This will create 3 class files in the same folder
7. Create the jar file:
   1. Wrap all the classes into a jar using command: jar cvf MaximumTemperature.jar \*.class
   2. This will create MaximumTemperature.jar in the same folder
8. Copy the sample input dataset file on to hadoop file system
   1. hadoop fs -copyFromLocal sampleinput.txt /myinput
   2. Make sure that myinput folder is created on HDFS file system
9. Run the jar file
   1. hadoop jar MaximumTemperature.jar MaximumTemperature sampleinput.txt /myoutput/
   2. If successful, an output file will be created in /myoutput folder by the name part-r-0000 (or equivalent)
   3. It will give maximum temperature value for each year

MaxTemperature.java

// MaxTemperature Application to find the maximum temperature in the weather dataset

// MaxTemperature

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 MaxTemperature {

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

if (args.length != 2) {

System.err.println("Usage: MaxTemperature <input path> <output path>");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(MaxTemperature.class);

job.setJobName("Max temperature");

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

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

job.setMapperClass(MaxTemperatureMapper.class);

job.setReducerClass(MaxTemperatureReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

}

}

// ^^ MaxTemperature

MaxTemperatureMapper.java

// cc MaxTemperatureMapper Mapper for maximum temperature example

// vv MaxTemperatureMapper

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 MaxTemperatureMapper

extends Mapper<LongWritable, Text, Text, IntWritable> {

private static final int MISSING = 9999;

@Override

public void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

String line = value.toString();

String year = line.substring(15, 19);

int airTemperature;

if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs

airTemperature = Integer.parseInt(line.substring(88, 92));

} else {

airTemperature = Integer.parseInt(line.substring(87, 92));

}

String quality = line.substring(92, 93);

if (airTemperature != MISSING && quality.matches("[01459]")) {

context.write(new Text(year), new IntWritable(airTemperature));

}

}

}

// ^^ MaxTemperatureMapper

MaxTemperatureReducer.java

// cc MaxTemperatureReducer Reducer for maximum temperature example

// vv MaxTemperatureReducer

**import** java.io.IOException;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.Text;

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

**public** **class** MaxTemperatureReducer

**extends** Reducer<Text, IntWritable, Text, IntWritable> {

@Override

**public** **void** reduce(Text key, Iterable<IntWritable> values,

Context context)

**throws** IOException, InterruptedException {

**int** maxValue = Integer.MIN\_VALUE;

**for** (IntWritable value : values) {

maxValue = Math.max(maxValue, value.get());

}

context.write(key, **new** IntWritable(maxValue));

}

}

// ^^ MaxTemperatureReducer

Portion of input file: MaxTemp.txt (Actual size=3.5 MB)

0161999999949882005010100536+42113-092918FM-15+0297KMIW V0203405N004152200059N0160935N5-00285-01005102515ADDAA101000095GA1005+999999999GD10991+9999999MA1102375098825REMMET09212/31/04 18:53:26 METAR KMIW 010053Z 34008KT 10SM CLR M03/M10 A3023 RMK AO2 SLP251 T10281100

0161999999949882005010101536+42113-092918FM-15+0297KMIW V0203605N003652200059N0160935N5-00395-00945102595ADDAA101000095GA1005+999999999GD10991+9999999MA1102475098925REMMET09212/31/04 19:53:26 METAR KMIW 010153Z 36007KT 10SM CLR M04/M09 A3026 RMK AO2 SLP259 T10391094

0181999999949882006010102536+42113-092918FM-15+0297KMIW V0200105N004152200059N0160935N5-00395-00945102635ADDAA101000095GA1005+999999999GD10991+9999999MA1102515098955MD1140224+9999REMMET09812/31/04 20:53:26 METAR KMIW 010253Z 01008KT 10SM CLR M04/M09 A3027 RMK AO2 SLP263 T10391094 51022

Output as generated and stored in part-r-00000 on hadoop file system:

2005 339

2006 28

2009 56