Big Data: Architectures and Data Analytics

July 14, 2017

Student ID _____

First Name _____

Last Name _____

The exam is **open book** and lasts **2 hours**.

Part I

Answer to the following questions. There is only one right answer for each question.

1. (2 points) Consider the HDFS files logs.txt and logs2.txt. The size of logs.txt is 1036MB and the size of log2.txt is 500MB. Suppose that the replication factor is 4 and the block size is 512MB? How many HDFS blocks are totally used to store the two files in HDFS?

    a) 3 blocks

    b) 4 blocks

    c) 12 blocks

    d) 16 blocks

2. (2 points) Consider the HDFS folder "inputData" containing the following three files:

Filename	Size	Content of the file
Temperature1.txt	61 bytes	2016/01/01,00:00,0 2016/01/01,00:05,-1 2016/01/01,00:10,-1.2
Temperature2.txt	63 bytes	2016/01/01,00:15,-1.5 2016/01/01,00:20,0 2016/01/01,00:25,-0.5
Temperature3.txt	62 bytes	2016/01/01,00:30,-0.5 2016/01/01,00:35,1 2016/01/01,00:40,1.5

Suppose that you are using a Hadoop cluster that can potentially run up to 10 mappers in parallel and suppose that the HDFS block size is 512MB.

Suppose that the following MapReduce program is executed by providing the folder "inputData" as input folder and the folder "results" as output folder.

```
/* Driver */
package it.polito.bigdata.hadoop.exam;

import ….;

public class DriverBigData extends Configured implements Tool {
        @Override
        public int run(String[] args) throws Exception {
                Path inputPath;
                Path outputDir;
                int exitCode;

                inputPath = new Path(args[0]);
                outputDir = new Path(args[1]);

                Configuration conf = this.getConf();

                Job job = Job.getInstance(conf);
                job.setJobName("Exercise #1 - Exam 2017/06/30");

                FileInputFormat.addInputPath(job, inputPath);
                FileOutputFormat.setOutputPath(job, outputDir);

                job.setJarByClass(DriverBigData.class);

                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                // Set mapper
                job.setMapperClass(MapperBigData.class);
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(DoubleWritable.class);

                // Set reduce class
                job.setReducerClass(ReducerBigData.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(DoubleWritable.class);

                // Set number of reducers
                job.setNumReduceTasks(2);

                // Execute the job and wait for completion
                if (job.waitForCompletion(true) == true)
                        exitCode = 0;
                else
                        exitCode = 1;

                return exitCode;
        }

        public static void main(String args[]) throws Exception {
                // Exploit the ToolRunner class to "configure" and run the Hadoop
                // application
                int res = ToolRunner.run(new Configuration(), new DriverBigData(), args);

                System.exit(res);
        }

}
```

```
/* Mapper */
package it.polito.bigdata.hadoop.exam;

import java.io.IOException;

import ….;

/* Mapper */
class MapperBigData extends Mapper<LongWritable, Text, Text, DoubleWritable> {

        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
                String fields[] = value.toString().split(",");
                String date = fields[0];
                Double temperature = Double.parseDouble(fields[2]);

                // Emit (date, temperature)
                context.write(new Text(date), new DoubleWritable(temperature));
        }
}

/* Reducer */
package it.polito.bigdata.hadoop.exam;

import …..;

class ReducerBigData extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
        @Override
        protected void reduce(Text key, // Input key type
                        Iterable<DoubleWritable> values, // Input value type
                        Context context) throws IOException, InterruptedException {

                double maxTemp = Double.MIN_VALUE;

                // Iterate over the set of values and compute the maximum temperature
                for (DoubleWritable temperature : values) {
                        if (temperature.get() > maxTemp) {
                                maxTemp = temperature.get();
                        }
                }

                // Emit (date, maximum temperature)
                context.write(key, new DoubleWritable(maxTemp));
        }
}
```

What is the content of the output folder generated by the execution of the application reported above?

a) The output folder contains three files

- One file that contains only the line "2016/01/01 0"

- One file that contains only the line "2016/01/01 0"

- One file that contains only the line "2016/01/01 1.5"

b) The output folder contains three files

- One file that contains only the line "2016/01/01 1.5"

- The other two files are empty

c) The output folder contains two files

- One file that contains only the line "2016/01/01 1.5"

- The other file is empty

d) The output folder contains two files

- One file that contains only the line "2016/01/01 0"

- One file that contains only the line "2016/01/01 1.5"

Part II

PoliWeather is an environmental company that monitors weather data for performing long-term analyses. Specifically, PoliWeather is focused on temperature analyses and the analyses of interest are based on the following data sets/files.

- Temperatures.txt
    - Temperatures.txt is a text file containing the historical information about the maximum and minimum daily temperatures on several European cities around the world. It contains the data about the last 20 years.
    - Each line of the input file has the following format
        - date_reading,city,country,max_temperature,min_temperature

            where *city* is a city name, country is the country of *city*, and *max_temperature* and *min_temperature* are the observed maximum and minimum temperatures in *city* at date *date_reading.*

        - For example, the line

            2016/07/20,Turin,Italy,32.5,26.0

            means that the observed maximum temperature in **Turin** on **July 20, 2016** was **32.5°C** and the observed minimum temperature in **Turin** on **July 20, 2016** was **26.0°C**

Exercise 1 – MapReduce and Hadoop (9 points)

The managers of PoliWeather are interested in selecting the cities associated with at least one day (date) characterized by a maximum temperature greater than 35°C and at least one day (date) characterized by a minimum temperature less than -20°C. All the historical data stored in Temperatures.txt must be considered.

Design a single application, based on MapReduce and Hadoop, and write the corresponding Java code, to address the following point:

A. *City with at least one high temperature value and at least one low temperature value*. Considering all the historical data stored in Temperatures.txt, the application must select the cities with at least one day (date) characterized by a maximum temperature greater than 35°C (i.e., *max_temperature* > 35°C) and at least one day (date) characterized by a minimum temperature less than -20°C (i.e., *min_temperature* < -20°C). The two days (dates) may be the same date or different dates. Store the results in a HDFS folder. The output contains one line for each of the selected cities.

The name of the output folder is one argument of the application. The other argument is the path of the input file Temperatures.txt.

Fill in the provided template for the Driver of this exercise. Use your papers for the other parts (Mapper and Reducer).

Exercise 2 – Spark and RDDs (18 points)

The managers of PoliWeather are interested in performing some analyses about the meteorological summer of year 2015. Specifically, for each city, you must compute the average value of *max_temperature* by considering all the values of *max_temperature* associated with the meteorological summer of year 2015 (i.e., the analysis is based on the historical data stored in Temperatures.txt, considering only the dates from June 1, 2015 to August 31, 2015).

PoliWeather is also interested in identifying the hot cities, for each country, during the meteorological summer of year 2015. Specifically, given a city, that city is classified as a "*hot city*" if the average value of *max_temperature* of that city is at least 5°C greater than the average value of *max_temperature* of the country of that city (i.e., the average value of *max_temperature* computed over all the *max_temperature* values of the country the city is part of). This analysis, about hot cities, must be performed only for the meteorological summer of year 2015 (i.e., the analysis is based on the historical data stored in Temperatures.txt, considering only the dates from June 1, 2015 to August 31, 2015).

The managers of PoliWeather asked you to develop an application to address the analyses they are interested in. The application has three arguments/parameters: the file Temperatures.txt and two output folders (associated with the outputs of the following points A and B, respectively).

Specifically, design a single application, based on Spark and RDDs, and write the corresponding Java code, to address the following points:

A. (8 points) *Meteorological summer 2015 - City average maximum temperature.* The application selects from Temperatures.txt only the historical temperature values observed from June 1, 2015 to August 31, 2015 and then computes, for each city, the *average value of max_temperature.* The application stores in the first HDFS output folder the information "(*city-country, average max_temperature of city)*" (note that each output key contains the concatenation of the city and its country). The output file contains one pair per line.

B. (10 points) *Meteorological summer 2015 - Hot cities.* Consider only the historical data of the meteorological summer of year 2015 (i.e., the analysis is based on the historical data stored in Temperatures.txt, considering only the dates from June 1, 2015 to August 31, 2015). The application stores in the second HDFS output folder only the "*hot cities*" of each country. Specifically, the hot cities are those with an average *max_temperature* value that is at least 5°C greater than the average *max_temperature* value of the country the city is part of.

Big Data: Architectures and Data Analytics

July 14, 2017

Student ID _____

First Name _____

Last Name _____

Use the following template for the Driver of Exercise 1

Fill in the missing parts. You can strikethrough the second job if you do not need it.

```
import ….
/* Driver class. */
public class DriverBigData extends Configured implements Tool {
        public int run(String[] args) throws Exception {
        Path inputPath = new Path(args[0]);
        Path outputDir = new Path(args[1]);
        Configuration conf = this.getConf();

        // First job
        Job job1 = Job.getInstance(conf);
        job1.setJobName("Exercise 1 - Job 1");
        // Job 1 - Input path
        FileInputFormat.addInputPath(job,_____);

        // Job 1 - Output path
        FileOutputFormat.setOutputPath(job,_____);

        // Job 1 - Driver class
        job1.setJarByClass(DriverBigData.class);

        // Job1 - Input format
        job1.setInputFormatClass(_____);

        // Job1 - Output format
        job1.setOutputFormatClass(_____);

        // Job 1 -  Mapper class
        job1.setMapperClass(Mapper1BigData.class);
        // Job 1 – Mapper: Output key and output value: data types/classes
        job1.setMapOutputKeyClass(_____);

        job1.setMapOutputValueClass(_____);

        // Job 1 -  Reducer class
        job.setReducerClass(Reducer1BigData.class);

        // Job 1 – Reducer: Output key and output value: data types/classes
        job1.setOutputKeyClass(_____);

        job1.setOutputValueClass(_____);

        // Job 1 - Number of reducers
        job1.setNumReduceTasks( 0[ _ ] or 1[ _ ] or >=1[ _ ] ); /* Select only one of the three options */
```

```java
                // Execute the first job and wait for completion
                if (job1.waitForCompletion(true)==true)
                {
                        // Second job
                        Job job2 = Job.getInstance(conf);
                        job2.setJobName("Exercise 1 - Job 2");
                        // Set path of the input folder of the second job
                        FileInputFormat.addInputPath(job2,_____);

                        // Set path of the output folder for the second job
                        FileOutputFormat.setOutputPath(job2,_____);

                        // Class of the Driver for this job
                        job2.setJarByClass(DriverBigData.class);

                        // Set input format
                        job2.setInputFormatClass(_____);

                        // Set output format
                        job2.setOutputFormatClass(_____);

                        // Set map class
                        job2.setMapperClass(Mapper2BigData.class);

                        // Set map output key and value classes
                        job2.setMapOutputKeyClass(_____);

                        job2.setMapOutputValueClass(_____);

                        // Set reduce class
                        job2.setReducerClass(Reducer2BigData.class);

                        // Set reduce output key and value classes
                        job2.setOutputKeyClass(_____);

                        job2.setOutputValueClass(_____);

                        // Set number of reducers of the second job
                        job2.setNumReduceTasks( 0[ _ ] or 1[ _ ] or >=1[ _ ] ); /*Select only one of the three
                                                                        options*/

                        // Execute the job and wait for completion
                        if (job2.waitForCompletion(true)==true)
                                exitCode=0;
                        else
                                exitCode=1;
                }
                else
                        exitCode=1;

                return exitCode;
        }
        /* Main of the driver  */
        public static void main(String args[]) throws Exception {
        int res = ToolRunner.run(new Configuration(), new DriverBigData(), args);
        System.exit(res);
        }
}
```