Big Data: Architectures and Data Analytics

June 26, 2018

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 folder logsFolder, which contains three files: log.txt , errors.txt, and warning.txt. The size of log.txt is 512MB, the size of errors.txt is 518MB, while the size of warning.txt is 250MB. Suppose that you are using a Hadoop cluster that can potentially run up to 9 mappers in parallel and suppose to execute a Map-only MapReduce-based program that selects only the lines of the input files in logsFolder containing the words “ERROR” or “WARNING”. The HDFS block size is 256MB. How many mappers are instantiated by Hadoop when you execute the application by specifying the folder logsFolder as input?

   a) 9

   b) 6

   c) 5

   d) 3

2. (2 points) Consider the HDFS folder “inputData” containing the following two files:

Filename	Size	Content of the files	HDFS Blocks	
			Block ID	Content of the block
Wind1.txt	18 bytes	11.00 19.00 40.00	B1	11.00 19.00
			B2	40.00
Wind2.txt	18 bytes	40.00 70.00 90.00	B3	40.00 70.00
			B4	90.00

   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 12 bytes.

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

```
/* Driver */
import … ;
public class DriverBigData extends Configured implements Tool {
	@Override
	public int run(String[] args) throws Exception {
		Configuration conf = this.getConf();
		Job job = Job.getInstance(conf);
		job.setJobName("2018/06/26 - Theory");

		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));

		job.setJarByClass(DriverBigData.class);

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

		job.setMapperClass(MapperBigData.class);
		job.setMapOutputKeyClass(DoubleWritable.class);
		job.setMapOutputValueClass(NullWritable.class);

		job.setNumReduceTasks(0);

		if (job.waitForCompletion(true) == true)
			return 0;
		else
			return 1;
	}

	public static void main(String args[]) throws Exception {
		int res = ToolRunner.run(new Configuration(), new DriverBigData(), args);
		System.exit(res);
	}
}

/* Mapper */
import …;

class MapperBigData extends Mapper<LongWritable, Text, DoubleWritable, NullWritable> {
	double sumWind;

	protected void setup(Context context) {
		sumWind = 0;
	}

	protected void map(LongWritable key, Text value, Context context) throws IOException,
	InterruptedException {
		double wind = Double.parseDouble(value.toString());
		sumWind = sumWind + wind;
	}

	protected void cleanup(Context context) throws IOException, InterruptedException {
		// emit the content of sumWind if it is greater than 45
		if (sumWind>45)
			context.write(new DoubleWritable(sumWind), NullWritable.get());
	}
}
```

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

a) The output folder contains four part files
   - One file that contains the following lines

     110

     90
   - Three empty files

b) The output folder contains four part files
   - One file that contains the following line

     30
   - One file that contains the following line

     40
   - One file that contains the following line

     110
   - One file that contains the following line

     90

c) The output folder contains four part files
   - One file that contains the following line

     110
   - One file that contains the following line

     90
   - Two empty files

d) The output folder contains four part files
   - One file that contains the following line

     240
   - Three empty files

Part II

PoliClimate is an Intergovernmental Agency focused on climate change. They have millions of sensors located around the world for monitoring precipitations and wind speed. To identify critical behaviors, PoliClimate computes a set of statistics about the climate conditions based on the following input file.

- ClimateData.txt
  - ClimateData.txt is a text file containing the historical information about precipitations and wind speed gathered by the sensors of PoliClimate. For each sensor, a new line is inserted in ClimateData.txt every 10 minutes. The number of managed sensors is more than 100,000 and ClimateData.txt contains the historical data about the last 5 years (i.e., ClimateData.txt contains more than 52 billion lines).
  - Each line of ClimateData.txt has the following format
    - SID,Timestamp,Precipitation_mm,WindSpeed

      where *SID* is the sensor identifier, *Timestamp* is the timestamp at which precipitation and wind speed are collected, *Precipitation_mm* is the measured precipitation (in mm) and *WindSpeed* is the measured wind speed (in km/h).

    - For example, the following line

      Sens2,2018/03/01,15:40,0.10,8.5

      means that sensor ***Sens2***, at ***15:40*** of ***March 1, 2018***, measured a precipitation equal to ***0.10*** mm and a wind speed equal to ***8.5*** km/h.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliClimate are interested in selecting the sensors that were frequently associated with low wind speed in the morning (i.e., from 8:00 to 11:59) in April 2018.

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

A. *Low wind speed in the morning of April 2018*. Considering only the measurements (i.e., lines of ClimateData.txt) associated with the time slot from 8:00 to 11:59 of April 2018, the application must select the identifiers (SID) of the sensors that were associated at least 1000 times with a WindSpeed value less than 1.0 km/h. Store the result of the analysis in a HDFS folder. The output file contains one line for each of the selected sensors. Specifically, each line of the output file contains the SID of one of the selected sensors.

The name of the output folder is one argument of the application. The other argument is the path of the input file ClimateData.txt. Note that ClimateData.txt contains the

measurements collected in the last 5 years but the analysis is focused only on April 2018 and the time slot from 8:00 to 11:59.

Fill out the provided template for the Driver of this exercise. Use your sheets of paper for the other parts (Mapper and Reducer).

Exercise 2 – Spark and RDDs (19 points)

The managers of PoliClimate are interested in performing some analyses about the average value of precipitations and wind speed for each sensor during the last two months (from April 2018 to May 2018) to identify critical hours.

The managers of PoliClimate are also interested in identifying sensors measuring an unbalanced daily wind speed during the last two months (from April 2018 to May 2018). Specifically, given a date and a sensor, the sensor measured an "*unbalanced daily wind speed"* during that date if that sensor, in that specific date, is characterized by at least 5 hours for which the minimum value of WindSpeed is less than 1 km/h and at least 5 hours for which the minimum value of WindSpeed is greater than 40 km/h.

The managers of PoliClimate asked you to develop one single application to address all the analyses they are interested in. The application has five arguments: precipitation threshold (*PrecThr*), wind speed threshold (*WindThr*), the input file ClimateData.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 write the corresponding Java code, to address the following points:

A. (8 points) *Critical (SID, hour) pairs in April - May 2018*. Considering only the measurements collected in the last two months (from April 2018 to May 2018), the application must select only the pairs (SID, hour) for which the average of Precipitation_mm for that pair is less than the specified threshold *PrecThr* and the average of WindSpeed for that pair is less than the specified threshold *WindThr* (*PrecThr* and *WindThr* are two arguments of the application). The application stores in the first HDFS output folder the information "SID-hour" for the selected pairs (SID, hour), one pair per line.

B. (11 points) *Pairs (SID,date) characterized by a daily unbalanced wind speed during the last two months*. Considering only the measurements collected in the last two months (from April 2018 to May 2018), the application must select the pairs (SID, date) that are characterized by an unbalanced daily wind speed. Specifically, given a date and a sensor, the sensor measured an "*unbalanced daily wind speed"* during that date if that sensor, in that specific date, is characterized by at least 5 hours for which the minimum value of WindSpeed is less than 1 km/h and at least 5 hours for which the minimum value of WindSpeed is greater than 40 km/h. The application stores in the second HDFS output folder the information "SID-date" for the selected pairs (SID, date), one pair per line.

Big Data: Architectures and Data Analytics

June 26, 2018

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 */
```

```
        // 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);
        }
}
```