Big Data: Architectures and Data Analytics

September 3, 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 files prices.txt and prices2.txt. The size of prices.txt is 1000MB and the size of prices2.txt is 536MB. Suppose that you are using a Hadoop cluster that can potentially run up to 10 mappers in parallel and suppose to execute a map-only MapReduce-based program that receives as input the folder containing prices.txt and prices2.txt and selects the rows of the two files containing prices less than 5.4. How many mappers are instantiated by Hadoop if the HDFS block size is 512MB?

   a) 10 mappers

   b) 4 mappers

   c) 3 mappers

   d) 2 mappers

2. (2 points) Consider the HDFS folder "inputFolder" containing the following two files:

Filename	Size	Content of the file
Temp1.txt	16 bytes	53.45 5.55 5.15
Temp2.txt	18 bytes	50.53 11.98 62.19

Suppose that you are using a Hadoop cluster that can potentially run up to 5 mappers in parallel and suppose that the HDFS block size is 1024MB.

Suppose that the following MapReduce program is executed by providing the folder "inputFolder" 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("2018/09/03 - 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 top1;

	protected void setup(Context context) {
		top1 = null;
	}

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

		Double val = new Double(value.toString());

		if (top1 == null || val.doubleValue() > top1) {
			top1 = val;
		}
```

```
        }

        protected void cleanup(Context context) throws IOException, InterruptedException {
                // emit the content of top1
                context.write(new DoubleWritable(top1), NullWritable.get());
        }
}
```

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

a) One file containing 62.19

b) Two files

- One containing the value 62.19
- One empty file

c) Two files

- One containing the value 53.45
- One containing the value 62.19

d) Two files

- One containing the value 53.45
- One containing the value 50.53

Part II

PoliTrades is an Italian company that collects and analyzes stock data. The managers of PoliTrades are interested in computing a set of statistics based on the following dataset file.

- Prices.txt
  - Prices.txt is a text file containing the historical information about the last 15 years of prices of thousands of stocks on several international financial markets.
  - The sampling rate is 2 minutes (i.e., every 2 minutes the system gathers the prices of the stocks under analyses and a new line for each stock is inserted at the end of Prices.txt)
  - Each line of the input file has the following format
    - date,hour:minute,stockId,price

      where *stockId* is a stock identifier (e.g., TSLA) and *price* is a floating point number whose value indicates the price of the stock *stockId* at time *date,hour:minute.*

    - For example, the line

      2014/04/01,10:02,TSLA,51.24

      means that the price of stock ***TSLA*** on ***April 1, 2014*** at ***10:02*** was ***51.24€***

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliTrades are interested in selecting only for the TSLA stock (stockId="TSLA") the lowest price reached by that stock during year 2014 and the last time stamp of year 2014 in which that lowest price has been reached.

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

A. *Lowest TSLA price in year 2014 and associated time stamp*. Considering only the prices of the TSLA stock (stockId="TSLA") in year 2014, the application must select the lowest price reached by the TSLA stock during year 2014 and the last time stamp (date+hour+minute) of year 2014 in which that price was reached. Store the result of the analysis in a HDFS folder. The output contains one single line with the selected lowest price and the associated last time stamp in the form

   date,hour:minute\t*price*

The arguments of the Hadoop application are (i) the path of the input file Prices.txt and (ii) the name of the output folder. Note that Prices.txt contains the prices associated with the

last 15 years and all stocks but the analysis we are interested in is limited to year 2014 and the TSLA stock only.

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 PoliTrades are interested in (A) performing some analyses about the daily variations of each stock during year 2014 and (B) identifying "unstable trends in year 2014" for each stock.

The managers of PoliTrades asked you to develop **one single application** to address all the analyses they are interested in. The application has the following arguments: the path of the input file Prices.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. *Analyses of daily variations in year 2014*. Considering only the prices in year 2014, the application must compute for each stock the number of dates associated with a daily price variation less than 5 euro. Given a stock and a date, the daily price variation of that stock in that date is given by the difference between the highest price and the lowest price reached by that stock in that specific date (highest price–lowest price). Store in the first HDFS output folder the set of "*stockId*,*number_of_dates_dailyVariation<5*", one line for each *stockId*, reporting **only** the stocks associated with **at least one date** in year 2014 with a daily price variation less than 5 euro.
   Sorting the results is not required.

B. *Unstable trends in year 2014 for each stock*. Considering only the prices in year 2014, for each stock, the application must select all the sequences of two consecutive dates characterized by an "*unstable trend*". Given a stock and two consecutive dates, the trend of that stock for those two dates is classified as an "*unstable trend*" if and only if the absolute difference between the daily price variations of the two dates is greater than 1 euro (i.e., abs(daily price variation of the first date - daily price variation of the second date) > 1). The application stores in the second HDFS output folder the full set of all unstable trends, one per line. Specifically, each line contains the following information about one unstable trend: stockId, first date of the unstable trend. Each output line has the following format:

   stockId,first_date

   For instance, suppose that stock TSLA has a daily variation equal to 2.03 in April 3, 2014 and a daily variation equal to 5.05 in April 4, 2014. It means that "TSLA,2014/04/03,2014/04/04" is classified as an "unstable trend" because abs(2.03-5.05)>1 and the following line is inserted in the output:

   TSLA,2014/04/03

   Note that you can have many pairs of consecutive dates of unstable trends for each stock, hence having multiple lines in the output file for each stock.

Sorting the results is not required.

Suppose that someone has already implemented the following static method.

- *public static String previousDate(String date)* of the *DateTool* class.
    - The parameter of this method is a string representing a date (in the format yyyy/mm/dd). The returned value is a string representing the previous date.
    - For example, the invocation

        String yesterday=DateTool.previousDate(“2016/06/20”);

      stores “2016/06/19” in the variable *yesterday.*

Big Data: Architectures and Data Analytics

July 16, 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 instances of the reducer of the first job
        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(_____);

                // Job 2 - Number instances of the reducer 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);
        }
}
```