Big Data: Architectures and Data Analytics

June 30, 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 524MB and the size of log2.txt is 500MB. Suppose that you are using a Hadoop cluster that can potentially run up to 100 mappers in parallel and suppose to execute a (map-only) MapReduce-based program that receives as input the folder containing logs.txt and logs2.txt and selects the rows of the two files containing the words “WARNING” or “ERROR”. How many mappers are instantiated by Hadoop if the HDFS block size is 512MB?

   a) 2 mappers

   b) 3 mappers

   c) 4 mappers

   d) 9 mappers

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

Filename	**Size**	**Content of the file**
Prices1.txt	17 bytes	50.45 10.45 9.45
Prices2.txt	18 bytes	10.53 10.99 54.99

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 */
```
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("2017/06/30 - 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 54.99

b) One file containing 9.45

c) Two files

- One containing the value 50.45
- One containing the value 54.99

d) Two files

- One containing the value 50.45
- One containing the value 10.53

Part II

PoliFinance is a company that monitors and analyzes financial data. Specifically, PoliFinance is focused on stock analyses. The analyses of interest are based on the following data sets/files.

- Prices.txt
  - Prices.txt is a text file containing the historical information about the prices of stocks on several financial markets.
  - The sampling rate is 5 minutes (i.e., every 5 minutes the system collects the prices of the stocks under analyses and a new line for each stock is inserted in Prices.txt)
  - Each line of the input file has the following format
    - stockId,date,hour:minute,price

      where *stockId* is a stock identifier, *price* is the price of stock stockId at time *date,hour:minute.*
    - For example, the line

      FCAU,2016/06/20,16:10,10.43

      means that the price of stock ***FCAU*** on ***June 20, 2016*** at ***16:10*** was ***10.43€***

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliFinance are interested in analyzing for each stock its behavior in the months of year 2016. Specifically, only for year 2016, they are interested in selecting the months associated with high variations of the stock prices.

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

A. *Monthly high price variations*. Only for the historical data of year 2016, the application must compute for each pair (stockId,month) the absolute difference between the highest and the lowest price of the stockId during the month and also the *monthly percentage price variation* (i.e., (highest price–lowest price)/lowest price). Store the results, in an HDFS folder, only for those pairs (stockId,month) characterized by a *monthly percentage price variation* greater than 5%. The output contains one line for each of the selected pairs (stockId,month), and the format of each output line is as follows

   stockid_month**\t**absolute-difference,monthly-percentage-price-variation

   e.g., FCAU_06   1.05,11

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

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

Exercise 2 – Spark and RDDs (19 points)

The managers of PoliFinance are interested in selecting the lowest price for each pair (stock, date) by considering only the historical data of year 2016. This result is exploited to draw a chart that is used to analyze the performances of the stocks.

PoliFinance is also interested in identifying the stocks that are frequently characterized by a "positive weekly trend". Specifically, given a week of the year and a stock, the weekly trend of the stock in that week is classified as a "positive weekly trend" if the difference between the highest stock price of the last day of the week and the highest stock price of the first day of the week is greater than 0. The application must select those stocks that are characterized by at least *NW* "positive weekly trends" (i.e., at least *NW* weeks with a positive trend for each of the selected stocks). *NW* is an integer number and is a parameter of the application. The analysis is based on the historical data stored in Prices.txt, considering only year 2016.

Note that you can easily compare dates by representing them as strings based on the format "year/month/day". For instance, the string "2016/06/19" precedes the string "2016/06/20" (and date 2016/06/19 precedes date 2016/06/20).

Suppose that someone has already implemented the following static method.

- *public static Integer weekNumber(String date)* of the *DateTool* class.
    - The parameter of this method is a string representing a date. The returned value is an integer value that uniquely identifies the week of the year to which the provided date belongs.
    - For example, the invocation

      Integer weekNum=DateTool.*weekNumber*(“*2016/06/20*”);

      stores 25 in the variable weekNum because 2016/06/20 is part of the 25th week of year 2016.

The managers of PoliFinance asked you to develop an application to address the analyses they are interested in. The application has four arguments/parameters: the file Prices.txt, the value of NW, 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. *Lowest stock price per pair (stock,date).* The application selects from Prices.txt only the historical prices associated with year 2016 and then computes, for each pair (stockId, date), the lowest price. The application stores in the first HDFS output folder the information “(*stockId_date,lowest price)*” for each pair (stockId, date). The results are stored in ascending order by considering the fields stockId, date (i.e., the results must be sorted by stockId; if the stockId is the same, then the date is considered).

B. *Select stocks that are frequently characterized by a “positive weekly trend”*. The application must count, for each stockId, the number of weeks with a *“positive weekly trend”,* based on the definition reported above, by considering only year 2016. Finally, the application stores in the second HDFS output folder only the stockIds of the stocks characterized by at least *NW* “positive weekly trends” (i.e., at least *NW* weeks with a positive trend). The output file contains one stockId per line.