Big Data: Architectures and Data Analytics

January 24, 2020

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 "inputFolder" containing the following two files:

Filename	Size	Content of the file
HumidityA.txt	16 bytes	51.45 9.55 8.15
HumidityB.txt	18 bytes	40.53 12.98 52.99

Suppose that you are using a Hadoop cluster that can potentially run up to 4 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("2020/01/24 - 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, top2;

	protected void setup(Context context) {
		top1 = null;
		top2 = 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) {
			top2 = top1;
			top1 = val;
		}
		else {
			if (top2 == null || val.doubleValue() > top2)
				top2 = val;
		}

	}

	protected void cleanup(Context context) throws IOException, InterruptedException {
		// emit the content of top1 and top2
		context.write(new DoubleWritable(top1), NullWritable.get());
		context.write(new DoubleWritable(top2), NullWritable.get());
	}
}
```

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

a) Four files

- One file containing four lines associated with the values 52.99, 51.45, 40.53 and 9.55 (one value per line)
- Three empty files

b) Four files

- One containing the value 52.99
- One containing the value 51.45
- Two empty files

c) Two files

- One containing two lines associated with the values 51.45 and 9.55 (one value per line)
- One containing two lines associated with the values 52.99 and 40.53 (one value per line)

d) Two files

- One file containing two lines associated with the values 52.99 and 51.45 (one value per line)
- One empty file

2. (2 points) Consider an input HDFS folder *salesFolder* containing the following two files: sales2017.txt and sales2015.txt. sales2017.txt contains the sales related to year 2017 and its size is 512MB while sales2015.txt contains the sales of year 2015 and its size is 258MB. Suppose that you are using a Hadoop cluster that can potentially run up to 10 instances of the mapper in parallel. Suppose to execute a map-only application, based on MapReduce, that selects only the lines containing the sales of year 2017, by specifying *salesFolder* as input folder. The HDFS block size is 256MB. How many mappers are instantiated by Hadoop when you execute the application by specifying the folder *salesFolder* as input?

a) 10

b) 4

c) 3

d) 2

Part II

PoliAds is an international advertising company that manages the publication of advertisements on the web. To improve the effectiveness of its advertisements, PoliAds computes a set of statistics about the managed items and the published advertisements based on the following input data sets/files.

- Items.txt
  - Items.txt is a textual file containing the information about the items that are managed by PoliAds. There is one line for each item and the total number of items is greater than 20,000,000 (i.e., Items.txt contains more than 20,000,000 lines)
  - Each line of Items.txt has the following format
    - ItemID,Name,Description,Company,url

      where *ItemID* is the item identifier, *Name* and *Description* are its name and brief description, respectively, *Company* is the name of the company that sells the item and *url* is the url of the web page containing a detailed description of the item.

    - For example, the following line

IID1,t-shirt-It,T-shirt with the Italian flag,MyFashionCompany,http://myfashion/tsit.html

means that item with id ***IID1*** is characterized by the name **t-shirt-it** and it is sold by **MyFashionCompany**. Moreover, its brief description is "***T-shirt with the Italian flag***" and the url of the web page associated with item ***IID1*** is ***http://myfashion/tsit.html***.

- Ads_Impressions.txt
  - Ads_Impressions.txt is a textual file containing the information about the visualizations of the web pages containing advertisements associated with items on the web. Each advertisement is associated with one single item. A new line is inserted in Ads_Impressions.txt every time a user visualizes a web page containing an advertisement. If the visualized web page contains more than one advertisement, and hence more than one item is advertised, one line per advertisement (i.e., item) is stored in Ads_Impressions.txt. Moreover, every time a web page containing advertisements is visualized, PoliAds also knows if the anonymous user who visualized that web page at that specific time stamp clicked on the visualized advertisements.

    Ads_Impressions.txt contains the historical data about the last 5 years.

  - Each line of Ads_Impressions.txt has the following format
    - ItemID,Timestamp,WebPage,Click

      where *Timestamp* is the time stamp at which the advertisement associated with item *ItemID* has been visualized on *WebPage*. *Click* is a Boolean variable that is equal to true if the anonymous user who

visualized that web page at time stamp *Timestamp* clicked on the advertisement associated with item *ItemID*. Otherwise, *Click* is set to false.

- For example, the following line

  IID1,*2019/01/01-08:15*,*www.polito.it/courses.html*,*true*

  means that at ***08:15*** of ***January 1, 2019*** an advertisement about item ***IID1*** has been visualized on the web page *www.polito.it/courses.html* and the user who visualized that page at that time stamp clicked on the advertisement associated with item ***IID1***.

  Note that the same item can be associated with several web pages in different time stamps or the same web pages in several time stamps. Moreover, the same item can be associated with the same web page at the same time stamp (many users can visualize the same web page at the same time stamp). For each visualization (i.e., for each triple *ItemID, Timestamp, WebPage*), one line is inserted in Ads_Impressions.txt. For example, if three users visualize the same web page at the same time stamp three lines are inserted in Ads_Impressions.txt (one for each user). In each line the value of *Click* depends on the action of the user associated with that specific visualization.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliAds are interested in analyzing the information about the number of clicks.

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

A. *Items with a high percentage of clicks in year 2019*. The application analyzes only the data related to year 2019 and selects the items with a percentage of clicks during year 2019 greater than 0.5%. The percentage of clicks for an item during year 2019 is given by 100*number of clicks associated with that item in year 2019/number of visualizations of that item in year 2019. Store the identifiers of the selected items in an HDFS folder. Each output line of the output contains one single item identifier (one of the selected items per line).

The name of the output folder is an argument of the application. The other argument is the path of the input file Ads_Impressions.txt, which contains the information about all the visualizations related to the last 5 years. Pay attention that we are interested only in the visualizations related to year 2019.

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 PoliAds are interested in performing some further analyses about the successfulness of the advertisements and the advertised items.

The managers of PoliAds asked you to develop one single application to address all the analyses they are interested in. The application has four arguments: the input files Items.txt and Ads_Impressions.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 (or Scala) code, to address the following points:

A. (9 points) *Increasing visualization trend.* The application selects the identifiers of the items that satisfy the two constraints reported in the following. Specifically, an item is selected if (i) the number of visualizations of that item during year 2017 is greater than the number of its visualizations during year 2016 and (ii) the number of clicks (Click=true) of that item during year 2017 is greater than the number of clicks (Click=true) of that item during year 2016. The application stores the identifiers of the selected items in the first HDFS output folder (one item identifier per line).

B. (10 points) *Most visualized item(s) during year 2018*. Considering only the visualizations related to year 2018 (i.e., the lines of Ads_Impressions.txt associated with year 2018), the application selects the name(s) of the most frequently visualized item(s) (i.e., the name(s) of the item(s) characterized by the maximum number of visualizations in year 2018). **Pay attention** that **many items can be associated with the maximum number of visualizations** during year 2018. The application stores in the second HDFS output folder the name(s) of the selected item(s) (one item per line). **Pay attention** that you must **store the name(s)** and not the item identifier(s).

Big Data: Architectures and Data Analytics

January 24, 2020

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 exactly 1[ _ ] or any number >=1[ _ ] ); /* Select only one of
                                                                                    these 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 of instances of the reducer of the second Job
                Job2.setNumReduceTasks( 0[ _ ] or exactly 1[ _ ] or any number >=1[ _ ] ); /* Select only
                                                                        one of these 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);
        }
}
```