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 min1, min 2;

	protected void setup(Context context) {
		min1 = null;
		min2 = null;
	}

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

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

		if (min1 == null || val.doubleValue() < min1) {
			min2 = min1;
			min1 = val;
		}
		else {
			if (min2 == null || val.doubleValue() < min2)
				min2 = val;
		}

	}

	protected void cleanup(Context context) throws IOException, InterruptedException {
		// emit the content of min1 and min2
		context.write(new DoubleWritable(min1), NullWritable.get());
		context.write(new DoubleWritable(min2), NullWritable.get());
	}
}
```

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

a) Two files

- One containing two lines associated with the values 8.15 and 9.55 (one value per line)
- One containing two lines associated with the values 12.98 and 40.53 (one value per line)

b) Two files

- One file containing two lines associated with the values 8.15, 9.55 (one value per line)
- One empty file

c) Four files

- One file containing four lines associated with the values 8.15, 9.55, 12.98 and 40.53 (one value per line)
- Three empty files

d) Four files

- One containing the value 8.15
- One containing the value 9.55
- Two empty files

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 258MB while sales2015.txt contains the sales of year 2015 and its size is 512MB. 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) 2

b) 3

c) 4

d) 10

Part II

PoliStores is an international company that sells items online. To improve the revenue of PoliStores, a set of statistics about the managed items are computed based on the following input data sets/files.

- Visualizations_Purchases.txt
  - Visualizations_Purchases.txt is a textual file containing the information about the visualizations of the web pages containing the description of the available items. Each web page is associated with one single item. A new line is inserted in Visualizations_Purchases.txt every time a user visualizes/watches the web of an item. Moreover, every time a user watches a web page of an item, PoliStores also knows if the user who watched that web page at that specific time stamp then purchased that item after the visualization.

    Visualizations_Purchases.txt contains the historical data about the last 10 years.

  - Each line of Visualizations_Purchases.txt has the following format
    - IID,Timestamp,Purchased,Username

      where *Timestamp* is the time stamp at which user *Username* watched the web page associated with item *IID*. *Purchased* is a Boolean variable that is equal to true if user *Username* purchased item *IID* after the visualization of the web page associated with item *IID*. Otherwise, *Purchased* is set to false.

    - For example, the following line

      IID1,2018/02/02-08:05,false,PaoloG

      means that at ***08:05*** of ***February 2, 2018***, user ***PaoloG*** watched the web page associated with item ***IID1*** and he did not purchase (buy) item ***IID1*** after the visualization of that web page.

      Note that many users can watch the same web page (i.e., the same item) at the same time stamp and the same user can watch the same web page (i.e., the same item) in different time stamps. For each visualization (i.e., for each triple *IID, Timestamp, Username*), one line is inserted in Visualizations_Purchases.txt. For example, if three users watch the same web page (i.e., the same item) at the same time stamp three lines are inserted in Visualizations_Purchases.txt (one for each user). In each line the value of *Purchased* depends on the action of the user associated with that specific visualization.

- Items.txt
  - Items.txt is a textual file containing the information about the items that are sold by PoliStores. There is one line for each item and the total number of

items is greater than 10,000,000 (i.e., Items.txt contains more than 10,000,000 lines)

- Each line of Items.txt has the following format
  - IID,Name,ShortDescription,Image

    where *IID* is the item identifier, *Name* and *ShortDescription* are its name and brief description, respectively, and *Image* is the url of an image associated with the item.

  - For example, the following line

    IID1,t-shirt-summer,Summer t-shirt,http://PoliStores.it/ID1.jpg

    means that item with id ***IID1*** is characterized by the name **t-shirt-summer**, its short description is "***Summer t-shirt***" and the url of the image associated with item ***IID1*** is ***http://PoliStores.it/ID1.jpg***.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliStores are interested in analyzing the information about the number of purchases.

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 purchases in April 2018*. The application analyzes only the data related to April 2018 and selects the items with a percentage of purchases in April 2018 greater than 0.1%. The percentage of purchases for an item in April 2018 is given by 100*number of purchases associated with that item in April 2018/number of visualizations of that item in April 2018. 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 Visualizations_Purchases.txt, which contains the information about all the visualizations related to the last 10 years. Pay attention that we are interested only in the visualizations related to April 2018.

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 PoliStores are interested in performing some further analyses about the available items.

The managers of PoliStores asked you to develop one single application to address all the analyses they are interested in. The application has four arguments: the input files Visualizations_Purchases.txt and Items.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 purchase 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 in March 2019 is greater than the number of its visualizations in February 2019 and (ii) the number of purchases (Purchased=true) of that item in March 2019 is greater than the number of purchases (Purchased=true) of that item in February 2019. The application stores the identifiers of the selected items in the first HDFS output folder (one item identifier per line).

B. (10 points) *Most purchased item(s) in June 2019*. Considering only the visualizations related to June 2019 (i.e., the lines of Visualizations_Purchases.txt associated with June 2019), the application selects the name(s) of the most frequently purchased item(s) (i.e., the name(s) of the item(s) characterized by the maximum number of purchases in June 2019). **Pay attention** that **many items can be associated with the maximum number of purchases** in June 2019. 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);
        }
}
```