Big Data: Architectures and Data Analytics

September 14, 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 folder logsFolder, which contains two files: logs.txt and logs2.txt. The size of logs.txt is 500MB and the size of logs2.txt is 524MB. Suppose that you are using a Hadoop cluster that can potentially run up to 10 mappers in parallel and suppose to execute a MapReduce-based program that selects the rows of the files in logsFolder containing the word "ERROR". Which of the following values is a proper HDFS block size if you want to "force" Hadoop to run exactly 2 mappers in parallel when you execute the application by specifying the folder logsFolder as input?

   a) Block size: 1024MB

   b) Block size: 512MB

   c) Block size: 256MB

   d) Block size: 128MB

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
Prices1.txt	18 bytes	21.45 52.55 43.55	B1	21.45 52.55
			B2	43.55
Prices2.txt	18 bytes	60.33 60.33 60.33	B3	60.33 60.33
			B4	60.33

Suppose that you are using a Hadoop cluster that can potentially run up to 20 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("2017/09/14 - 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 firstPrice;

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

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

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

		if (firstPrice == null || price.doubleValue() > firstPrice) {
			firstPrice = price;
		}

	}

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

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

a) The output folder contains two files

- One file that contains the following line

  52.55

- One file that contains the following line

  60.33

b) The output folder contains four files

- One file that contains the following line

  52.55

- One file that contains the following line

  43.55

- One file that contains the following line

  60.33

- A fourth file that contains the same content of the previous file, i.e., the following line

  60.33

c) The output folder contains only one file

- One file that contains the following four lines

  52.55

  43.55

  60.33

  60.33

d) The output folder contains only one file

- The output file contains the following line

  60.33

Part II

PoliFly is a big travel web site that allows booking flights. In order to identify critical airports and routes, PoliFly computes a set of statistics about flights and their delays based on the following input data sets/files.

- Flights.txt
  - Flights.txt is a text file containing the historical information about the flights of the airlines managed by PoliFly. The number of flights per day is more than 100,000 and Flights.txt contains the historical data about the last 10 years.
  - Each line of the input file has the following format
    - Flight_number,Airline,date,scheduled_departure_time,scheduled_arrival_time,departure_airport_id,arrival_airport_id,delay,cancelled,number_of_seats,number_of_booked_seats

      where *Flight_number* is the identifier of the flight, *Airline* is the airline that operated the flight, *date* is the date of the flight, *scheduled_departure_time* and *scheduled_arrival_time* are its scheduled departure and arrival times, departure_airport_id and arrival_airport_id are the identifiers of the departure and arrival airports, *delay* is the delay in minutes of the flight with respect to the *scheduled_arrival_time*, and *cancelled* is a flag that is 'yes' if the flight has been cancelled and 'no' otherwise. *number_of_seats* is the total amount of seats of the flight while *number_of_booked_seats* is the number of booked seats.

    - For example, the line

      AI1103,Alitalia,2016/03/01,15:35,17:10,TRN,CDG,5,no,150,143

      means that the flight ***AI1103***, operated by ***Alitalia***, from ***TRN*** to ***CDG*** scheduled for ***March 1, 2016***, scheduled departure time ***15:35*** - scheduled arrival time ***17:10***, arrived at the CDG airport ***5*** minutes late. The flight had ***150*** seats and only ***143*** were booked.
- Airports.txt
  - Airports.txt is a text file containing the information about airports. Each line contains the information about one airport.
  - Each line of Airports.txt has the following format
    - Airport_id,Airport_name,City,Country

      where *Airport_id* is the identifier of the airport, *Airport_name* is its name, and *city* and *country* are the city and the country where the airport is located, respectively.
    - For example, the line

      TRN,Torino Caselle,Caselle Torinese,Italy

means that ***TRN*** is the id of the ***Torino Caselle*** airport, which is located in ***Caselle Torinese***, ***Italy***.

Exercise 1 – MapReduce and Hadoop (9 points)

The managers of PoliFly are interested in selecting the airports characterized by more than 1% cancelled departing flights in the last year (i.e., from September 1, 2016 to August 31, 2017).

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

A. *Airports with many cancelled flights*. Considering only the subset of flights of the last year, the application must select the ids of the airports with more than 1% cancelled departing flights in the last year (i.e., from September 1, 2016 to August 31, 2017). The percentage of cancelled departing flights for an airport is given by the ratio between the number of cancelled flights departing from that airport and the total number of flights departing from that airport. Store the result of the analysis in a HDFS folder. The output file contains one line for each of the selected airports. Each line of the output file has the following format

    - departure_airport_id\t*Percentage of cancelled flights*

The name of the output folder is one argument of the application. The other argument is the path of the input file Flights.txt. Note that Flights.txt contains the data of the last 10 years but the analysis is focused only on the flights of the last year (i.e., from September 1, 2016 to August 31, 2017).

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

Exercise 2 – Spark and RDDs (18 points)

The managers of PoliFly are interested in performing some analyses about the amount of delayed flights for each airline and each arrival airport, by considering only the flights with an arrival airport located in Germany. Specifically, they are interested in counting, for each couple (airline, arrival airport), with arrival airport located in Germany, the number of flights that arrived at least 15 minutes late and sorting the results based on the number of delayed flights.

Another analysis of interest is related to the identification of the “overloaded routes” between couples of airports. Each couple of airports (departure airport, arrival airport) is a route and a route is an “overloaded route” if at least 99% of the flights of that route were fully booked and at least 5% of the flights of that route were cancelled. A flight is fully booked if all the seats are booked (i.e., *number_of_booked_seats* == *number_of_seats*).

The managers of PoliFly asked you to develop an application to address all the analyses they are interested in. The application has four arguments/parameters: the files Flights.txt and Airports.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 RDDs, and write the corresponding Java code, to address the following points:

A. (9 points) *Airlines with delayed flights landing in Germany*. The application must select only the flights with an arrival airport located in Germany and then computes, for each couple (airline, arrival airport), the number of flights that arrived at least 15 minutes late. The application stores in the first HDFS output folder the information "number of delayed flights, airline, arrival airport name". The results are stored in decreasing order by considering the number of delayed flights. The output contains one couple "(airline, arrival airport name)", and the associated "number of delayed flights", per line. Note that the application stores the name of the arrival airport. You can suppose that the arrival airport name is unique (i.e., no airports have the same name).

B. (9 points) *Overloaded routes*. The application must select the "overloaded routes". Every couple of airport ids "(departure_airport_id,arrival_airport_id)" associated with at least one flight is a route[1]. A route is an "overloaded route" if at least 99% of the flights of that route were fully booked, based on the historical data available in Flights.txt, and at least 5% of the flights of that route were cancelled, based on the historical data available in Flights.txt. A flight is fully booked if all the seats are booked (i.e., *number_of_booked_seats* == *number_of_seats*). The application stores in the second HDFS output folder the information "(departure_airport_id,arrival_airport_id)" for the selected routes. Note that the application stores couples of airport ids and not their names. The output contains one line per selected route.

[1] Note that the "direction" is important. For instance, the couple of airport ids (TRN, CDG) is a route and the couple (DCG, TRN) is a different route.

Big Data: Architectures and Data Analytics

September 14, 2017

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);
        }
}
```