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 "inputData" containing the following two files:

Filename	Size	Content of the files	HDFS Blocks	
			Block ID	Content of the block
Mark1.txt	9 bytes	21 28 24	B1	21 28
			B2	24
Mark2.txt	9 bytes	30 30 30	B3	30 30
			B4	30

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 6 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 minimumMark;

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

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

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

		if (minimumMark == null || mark.doubleValue() < minimumMark) {
			minimumMark = mark;
		}

	}

	protected void cleanup(Context context) throws IOException, InterruptedException {
		// emit the content of minimumMark
		context.write(new DoubleWritable(minimumMark), 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

  21

- One file that contains the following line

  30

b) The output folder contains only one file

- The output file contains the following line

  21

c) The output folder contains four files

- One file that contains the following line

  21

- One file that contains the following line

  24

- One file that contains the following line

  30

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

  30

d) The output folder contains only one file

- One file that contains the following four lines

  21

  24

  30

  30

2. (2 points) Consider the HDFS folder logsFolder, which contains two files: logs.txt and logs2.txt. The size of logs.txt is 1048MB and the size of logs2.txt is 1000MB. Suppose that you are using a Hadoop cluster that can potentially run up to 20 mappers in parallel and suppose to execute a MapReduce-based program that selects the rows of the files in logsFolder containing the word “WARNING”. 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: 256MB

   b) Block size: 512MB

   c) Block size: 1024MB

   d) Block size: 2048MB

Part II

PoliTravel is a web site that aggregates data of several booking services and allows booking flights. To suggest reliable flights, PoliTravel computes a set of statistics that are used to characterize routes and airports based on number of cancelled flights and delays. The analyses are based on the following input data sets/files.

- 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
    - AirportID,City,Country,AirportName

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

      CDG,Paris,France,Charles de Gaulle

      means that ***CDG*** is the id of the ***Charles de Gaulle*** airport, which is located in ***Paris, France***.
- Flights.txt
  - Flights.txt is a text file containing the historical information about the flights of the airlines managed by PoliTravel. The number of flights per day is more than 50,000 and Flights.txt contains the historical data about the last 15 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

      LH1103,Lufthansa,2016/06/02,15:35,17:10,CDG,TRN,15,no,120,101 means that the flight ***LH1103***, operated by ***Lufthansa***, from ***CDG*** to ***TRN*** scheduled for ***June 2, 2016***, scheduled departure time ***15:35*** - scheduled arrival time ***17:10***, arrived at the TRN airport ***15*** minutes late. The flight had ***120*** seats and only ***101*** were booked.

Exercise 1 – MapReduce and Hadoop (9 points)

The managers of PoliTravel are interested in selecting the airports characterized by many late arriving flights in year 2016 (i.e., from January 1, 2016 to December 31, 2016). Specifically, the airports with more than 5% of the landing flights arriving at least 15 minutes late must be selected by the application and stored in the output HDFS folder.

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

A. *Destination airports with too many delayed flights in year 2016*. Considering only the subset of historical data/flights from January 1, 2016 to December 31, 2016, the application must select the ids of the airports with more than 5% of the flights arriving at least 15 minutes late in year 2016 (i.e., from January 1, 2016 to December 31, 2016). The percentage of flights arriving late for an airport is given by the ratio between the number of flights arriving at least 15 minutes late at that airport and the total number of flights arriving at 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

   - arrival_airport_id***\t***Percentage of delayed 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. Pay attention that Flights.txt contains the data of the last 15 years but the analysis is focused only on the flights of year 2016.

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 PoliTravel are interested in analyzing the amount of cancelled flights for each airline depending on the departure airport by considering only the flights with a departure airport located in France. Specifically, they are interested in counting, for each couple (airline, departure airport), with departure airport located in France, the number of cancelled flights and sorting the results based on the number of cancelled flights.

Another analysis of interest is related to the identification of the “underused routes” between couples of airports. Each couple of airports (departure airport, arrival airport) is a route and a route is an “underused route” if at least 25% of the flights of that route have at least 30% of not booked seats and at least 10% of the flights of that route were cancelled. The percentage of not booked seats is given by

100*(*number_of_seats -number_of_booked_seats)/number_of_seats*

The managers of PoliTravel asked you to develop an application to address all the analyses they are interested in. The application has 4 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 many cancelled flights departing from France*. The application must select only the flights with a departure airport located in France and then computes, for each couple (departure airport, airline), the number of cancelled flights. The application stores in the first HDFS output folder the information "number of cancelled flights, departure airport name, airline". The results are stored in decreasing order by considering the number of cancelled flights. The output contains one couple "(departure airport name, airline)", and the associated "number of cancelled flights", per line. Note that the application stores the name of the departure airport. You can suppose that the departure airport name is unique (i.e., there are not two airports with the same name).

B. (9 points) *Underused routes*. The application must select the "undersused 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 "underused route" if at least 25% of the flights of that route have at least 30% of not booked seats and at least 10% of the flights of that route were cancelled, based on the historical data available in Flights.txt. The percentage of not booked seats is given by "*100*(number_of_seats - 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);
        }
}
```