Big Data: Architectures and Data Analytics

September 19, 2019

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 cars.txt and cars2.txt. The size of cars.txt is 1028MB and the size of cars2.txt is 1020MB. Suppose that you are using a Hadoop cluster that can potentially run up to 10 mappers in parallel and suppose to execute a map-only MapReduce-based program that receives as input the folder containing cars.txt and cars2.txt and selects the rows of the two files containing the string "FIAT". How many mappers are instantiated by Hadoop if the HDFS block size is 1024MB?

   a) 2 mappers

   b) 3 mappers

   c) 4 mappers

   d) 10 mappers

2. (2 points) Consider the HDFS folder "inputFolder" containing the following two files:

Filename	Size	Content of the file
Temp1.txt	16 bytes	61.4 19.5 10.53
Temp2.txt	18 bytes	10.53 21.82 20.99

   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 2048MB.

   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("2019/09/19 - 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 lowestTemp;

	protected void setup(Context context) {
		lowestTemp = Double.MAX_VALUE;
	}

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

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

		if (val.doubleValue() < lowestTemp) {
			lowestTemp = val;
		}
	}

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

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

a) One file containing 10.53

b) Two files

- A first file containing the value 10.53
- A second file containing the value 10.53

c) Two files

- A first file containing the value 10.53
- A second file that is empty

d) Two files

- One containing the value 10.53
- One containing the value 19.5

Part II

PoliCar is an international free-floating car sharing company that manages cars around the world. To identify critical situations about its cars, PoliCar computes statistics about the managed cars and its customers based on the following input data sets/files.

- Cars.txt
  - Cars.txt is a textual file containing the information about the cars managed by PoliCar. There is one line for each car and the total number of cars is greater than 20,000 (i.e., Cars.txt contains more than 20,000 lines)
  - Each line of Cars.txt has the following format
    - CarID, Manufacturer

      where CarID is the car identifier and *Manufacturer* is the name of the company that manufactured the car.

    - For example, the following line

      C156,FIAT

      means that the car with id ***C156*** was manufactured by the ***FIAT*** manufacturer.

- Users.txt
  - Users.txt is a textual file containing the information about the users (customers) of PoliCar. There is one line for each user and the total number of users is greater than 1,000,000 (i.e., Users.txt contains more than 1,000,000 lines)
  - Each line of Users.txt has the following format
    - UserID,Name,Surname

      where UserID is the user identifier, while *Name* and *Surname* are his/her name and surname, respectively.

    - For example, the following line

      U100,John,Smith

      means that there is a user with id ***U100*** and his name and surname are ***John*** and ***Smith***, respectively.

- Cars_Trips.txt
  - Cars_Trips.txt is a textual file containing the information about the trips of the cars managed by PoliCar. Cars_Trips.txt contains the historical data about the last 5 years. Each line contains the data of one single trip. A new line is inserted in Cars_Trips.txt every time a new trip ends.
  - Each line of Cars_Trips.txt has the following format
    - City,Start-Timestamp,Duration,CarID,UserID

where *City* is the city associated with the trip (each trip is completely included in one single city), *Start-Timestamp* is the timestamp associated with the start of the trip and *Duration* is the duration of the trip in minutes. Finally, *CarID* is the identifier of the used car and *UserId* is the identifier of the user who used the car for the trip.

- For example, the following line

  Turin,2018/03/01_15:40,5,C156,U100

  means that user ***U100*** used car ***C156*** for a trip in ***Turin*** that started at ***15:40*** of ***March 1, 2018*** and lasted 5 minutes.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliCar are interested in analyzing the information about their users and understanding who used their car sharing service in more than one city.

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

A. *Users who used cars in at least two different cities*. The application analyzes all the trips related to year 2018 (Start-Timestamp in year 2018) and selects the UserID of the users who used the car sharing service in at least two different cities. Store the UserIDs of the selected users in an HDFS folder. The output contains one line per UserID.

   For example, suppose that

   - User U102 is associated with trips in Milan and Rome
   - User U104 is associated only with trips in Turin
   - User U105 is associated with trips in Milan, Rome, and Naples.

   Users U102 and U105 are selected because they used the car sharing service in at least two cities while U104 is discarded because he/she used the car sharing system in one single city.

The name of the output folder is an argument of the application. The other argument is the path of the input file Cars_Trips.txt, which contains the information about all trips. Pay attention you must focus your analysis only on the trips associated with year 2018 (Start-Timestamp in year 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 PoliCar are interested in performing some analyses about the trips of the cars in year 2018.

The managers of PoliCar asked you to develop one single application to address all the analyses they are interested in. The application has four arguments: the input files Cars.txt and Cars_Trips.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) *Cars constantly used during year 2018 in Turin.* Considering only the trips in Turin (City = Turin) during year 2018 (Start-Timestamp in year 2018), the application selects the cars that were frequently used in Turin in each month of year 2018. Specifically, a car is selected if it were used in at least 1000 trips in Turin during each month of year 2018. The application stores in the second HDFS output folder the CarIDs of the selected cars (one CarID per line).

   For instance, suppose that
   - Car156 is associated with the following number of trips in Turin during the months of year 2018:

Month	1	2	3	4	5	6	7	8	9	10	11	12
#Trips	**2000**	**1501**	**2021**	**1005**	**1050**	**1020**	**2034**	**1400**	**1560**	**1001**	**1400**	**1000**

   - Car254 is associated with the following number of trips in Turin during the months of year 2018: year 2018:

Month	1	2	3	4	5	6	7	8	9	10	11	12
#Trips	**2000**	**1501**	**2021**	**1005**	**950**	**900**	**1024**	**1400**	**1560**	**1001**	**1400**	**567**

   Car156 will be included in the output folder because it is associated with at least 1000 trips in Turin in each month of year 2018.

   Car254 will be discarded and **will not be stored** in the output folder because in some months of year 2018 (e.g., in June) there are less than 1000 trips in Turin associated with **Car254**.

B. (10 points) *Cars frequently used in Turin during year 2018*. Considering only the trips in Turin (City = Turin) during year 2018 (Start-Timestamp in year 2018), the application selects the cars that were used in more than 1% of the total number of trips in Turin during year 2018. The application stores in the first HDFS output folder for each selected car its **CarID** and **manufacturer** (one CarID and its manufacturer per line).

   For instance, suppose that the total number of trips in Turin (i.e., the trips with City = Turin) during year 2018 is 100,000 and (i) the car with CarID C201 was used in 20,000 trips in Turin during year 2018, while (ii) the car with CarID C502 was used in 500 trips in Turin during year 2018. The application selects car C201 (and its manufacturer) because 20,000 is greater than 1,000 (=100,000x0.01) while it discards car C502 because 500 is less than 1,000.

Big Data: Architectures and Data Analytics

September 19, 2019

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);
        }
}
```