Big Data: Architectures and Data Analytics

July 2, 2019

Student ID _____

First Name _____

Last Name _____

Part I

Answer to the following questions. There is only one right answer for each question.

2. (2 points) Consider the following driver of a Spark application.

```
package …  import ….
public class SparkDriver {
    public static void main(String[] args) {
        SparkConf conf=new SparkConf().setAppName("Exam");
        JavaSparkContext sc = new JavaSparkContext(conf);

        /* Analyze logs */
        JavaRDD<String> logsRDD = sc.textFile("logs.txt");
        Long numLines = logsRDD.count();
        System.out.println(numLines);

        /* Select URLs from logs */
        JavaRDD<String> URLsRDD = logsRDD.map(line -> {
                                    String[] fields = line.split(“,”);
                                    String URL = fields[0];
                                    return URL;
                                  });

        JavaRDD<String> distinctURLs = URLsRDD.distinct();
        Long numDistinctURLs = distinctURLs.count();

        System.out.println(numDistinctURLs);
        sc.close();
    }
}
```

Which one of the following statements is true?

a) Caching *logsRDD* can improve the efficiency of the application (in terms of execution time)

b) Caching *URLsRDD* can improve the efficiency of the application (in terms of execution time)

c) Caching *distinctURLs* can improve the efficiency of the application (in terms of execution time)

d) Caching the RDDs of this Spark application does not improve its efficiency (in terms of execution time)

2. (2 points) Consider an input HDFS folder *logsFolder* containing the files log2017.txt and log2018.txt. log2017.txt contains the logs of year 2017 and its size is 254MB while log2018.txt contains the logs of year 2018 and its size is 2050MB. 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 the lines containing the string 2017, by specifying *logsFolder* 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 *logsFolder* as input?

   a) 11

   b) 10

   c) 9

   d) 1

Part II

PoliBike is an international bike sharing company that manages bicycles around the world. To identify critical situations about its bicycles, PoliBike computes a set of statistics about the managed bicycles based on the following input data sets/files.

- Bicycles.txt
  - Bicycles.txt is a textual file containing the information about the bicycles managed by PoliBike. There is one line for each bicycle and the total number of bicycles is greater than 1,000,000 (i.e., Bicycles.txt contains more than 1,000,000 lines)
  - Each line of Bicycles.txt has the following format
    - BID,BicycleManufacturer,City,Country

      where *BID* is the bicycle identifier, *BicycleManufacturer* is the name of the company that manufactured the bicycle, *City* is the city in which the bicycle is used (each bicycle is used on one single city) and *Country* is the country of *City*.

    - For example, the following line

      BID13,BianchiCompany,Turin,Italy

means that the bicycle with id ***BID13*** was manufactured by ***BianchiCompany*** and it is used in ***Turin***, which is located in ***Italy***.

- Bicycles_Failures.txt
  - Bicycles_Failures.txt is a textual file containing the information about the component failures of the bicycles managed by PoliBike. Bicycles_ Failures.txt contains the historical data about the last 10 years. A new line is inserted in Bicycles_ Failures.txt every time a bicycle failure occurs.
  - Each line of Bicycles_ Failures.txt has the following format
    - Timestamp,BID,Component

      where *Timestamp* is the timestamp at which the bicycle with id BID had a failure of component *Component*.

    - For example, the following line

      *2018/03/01_15:40,BID13,*w*heel*

      means that at ***15:40*** of ***March 1, 2018*** the ***wheel*** component of bicycle ***BID13*** broken down.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliBike are interested in analyzing the information about the manufactures of the bicycles used in Italy.

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

A. *Italian cities with single manufacturer*. The application analyzes the Italian cities. Specifically, an Italian city is selected if all the bicycles associates with that city are characterized by the same manufacturer. Store the selected cities, and the associated manufactures, in an HDFS folder. Each output line of the output contains one pair (city,manufacturer), one line per city.

For example, suppose that all the bicycles associated with Turin are manufactured by BianchiCompany while among the bicycles associated with Milan some bicycles are manufactured by RossiCompany, some by BianchiCompany and some others by VerdiCompany. Turin is selected by the application and *Turin,BianchiCompany* is stored in the output folder while Milan is not selected.

The name of the output folder is an argument of the application. The other argument is the path of the input file Bicycles.txt, which contains the information about all the bicycles used around the world but pay attention that the analysis we are interested in is focused only on the bicycles used in Italy.

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 PoliBike are interested in performing some analyses about the failures of the bicycles in the last year.

The managers of PoliBike asked you to develop one single application to address all the analyses they are interested in. The application has four arguments: the two input files Bicycles.txt and Bicycles_Failures.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) *Critical bicycles during year 2018*. Considering only the failures related to year 2018, the application selects the bicycles with more than two wheel failures in at least one month of year 2018 (i.e., a bicycle is selected if there is at least one month in year 2018 for which that bicycle is associated with more than two failures of the wheel component). The application stores in the first HDFS output folder the BIDs of the selected bicycles (one BID per line).

B. (10 points) *Cities characterized by bicycles with few failures during year 2018.* Considering only the failures related to year 2018, the application selects the cities for which all bicycles are characterized by few failures. Specifically, a city is selected if all the bicycles of that city are associated with at most 20 failures during year 2018 (at most 20 failures for each bicycle). The application prints on the standard output the number of selected cities and stores in the second HDFS output folder the selected cities (one city per line).

   For instance,

   - Suppose that in Turin there are two bicycles and suppose that during year 2018 there were 6 failures for the first bicycle and 30 failures for the second bicycle. Turin **must not be** selected by the Spark application (Point B) because the second bicycle is associated with more than 20 failures.

   - Suppose that in Milan there are three bicycles and suppose that during year 2018 there were 5 failures for the first bicycle, 20 failures for the second bicycle, and 8 for the third bicycle. Milan **must be** selected by the Spark application (Point B) because all its bicycles are associated with at most 20 failures during year 2018.

   Note. Pay attention that there are bicycles without failures (i.e., some bicycles occur in *Bicycles.txt* but not in *Bicycles_Failures.txt*).

Big Data: Architectures and Data Analytics

July 2, 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);
        }
}
```