Big Data: Architectures and Data Analytics

February 15, 2019

Student ID _____

First Name _____

Last Name _____

Part I

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

1. (2 points) Consider the HDFS file logs.txt. The size of logs.txt is 5000MB. 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 logs.txt containing the word “ERROR” or “WARNING”. Which of the following values is a proper HDFS block size if you want to “force” Hadoop to run 10 mappers in parallel when you execute the application by specifying logs.txt as input file?

   a) Block size: 5000MB

   b) Block size: 2048MB

   c) Block size: 1024MB

   d) Block size: 512MB

2. (2 points) Consider the cache “mechanism” of Spark. Which one of the following statements is true?

   a) Caching an RDD that is generated from another RDD is never useful

   b) At most one RDD per application can be cached

   c) Caching an RDD that is used multiple times in an application could improve the efficiency of the application (in terms of execution time)

   d) An RDD must never be cached

Part II

PoliTravel is an international online travel agency operating all around the world. To plan better travels for their clients, the managers of the PoliTravel are interested in characterizing the cities around the world based on the amount and types of POIs (points of interest) of each city. Each city can have thousands of POIs.

The analyses are based on the following input data set/file.

- POIs.txt
  - POIs.txt is a text file containing the information about the POIs of all the cities of the world (i.e., billions of POIs).
  - Each line of POIs.txt contains the information about one POI and has the following format
    - POI_ID,latitude,longitude,city,country,category,subcategory

      where *POI_ID* is the identifier of the represented POI, *latitude* and *longitude* represent the geographical position of that POI, *city* and *country* are the city and the country associated with that POI, respectively, and *category* and *subcategory* represent its category and subcategory.

    - For example, the following line

      P101,45.0621644,7.578633,Turin,Italy,shop,shoes

      means that POI ***P101*** represents a ***shop*** (category) of ***shoes*** (subcategory) that is located in ***Turin*** (***Italy***), and its geographical position is (lat=***45.0621644***, long=***7.578633***).

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliTravel are interested in selecting the subset of Italian cities with many "tourism" POIs, several of which are museums.

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

A. *Italian cities with many "turism" POIs and several museums*. Considering only the Italian cities and their POIs, the application must select the Italian cities with (i) more than 1000 tourism POIs (category="tourism") and (ii) at least 20 museums (subcategory="museum"). Note that the "museum" subcategory is a subcategory of the "tourism" category. The output file contains one line for each of the selected Italian cities (one city name per line).

The name of the output folder is an argument of the application. The other argument is the path of the input file POIs.txt. Note that POIs.txt contains the POIs of all the cities around the world but the analysis is focused only on the Italian cities.

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 management of PoliTravel is interested in selecting the Italian cities with taxis but without buses (i.e., those with POIs with subcategory="Taxi" but without POIs with subcategory="Busstop").

Moreover, the managers of PoliTravel are interested in selecting the Italian cities with a number of museums greater than the average number of museums per city in Italy.

The managers of PoliTravel asked you to develop one single application to address the two analyses they are interested in. The application has three arguments: the input file POIs.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 code, to address the following points:

A. (8 points) *Italian cities with taxis but without buses*. Considering only the Italian cities, the application must select the Italian cities with at least one "taxi" POI (subcategory="Taxi") but without "Busstop" POIs (subcategory="Busstop"). The application stores in the first HDFS output folder the selected cities, one city per line.

B. (11 points) *Italian cities with "many" museums with respect to the other Italian cities.* Considering only the Italian cities, the application must select the Italian cities with a number of "museum" POIs (subcategory="museum") greater than the average number of "museum" POIs per city in Italy. The application stores in the second HDFS output folder the selected cities, one city per line.

   For instance, as a running example, suppose that in Italy there are only three cities: Carmagnola, Rome, and Naples. Suppose that Carmagnola has 0 "museum" POIs, Rome has 42 "museum" POIs, and Naples has 18 "museum" POIs. Hence, the average number of "museum" POIs per city in Italy is 20 and only Rome must be selected.

Big Data: Architectures and Data Analytics

February 15, 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 reducers
        job1.setNumReduceTasks( 0[ _ ] or exactly 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 exactly 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);
        }
}
```