Big Data: Architectures and Data Analytics

September 17, 2020

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 input HDFS folder *myFolder* that contains the following two files:

    - ProfilesItaly.txt
        - the text file ProfilesItaly.txt contains the following three lines (size: 37 bytes)
          Paolo,Turin
          Luca,Turin
          Giovanni,Turin
    - ProfilesFrance.txt
        - the text file ProfilesFrance.txt contains the following two lines (size: 24 bytes)
          Paolo,Nice
          Luis,Paris

    Suppose that you are using a Hadoop cluster that can potentially run up to 10 instances of the mapper class in parallel. Suppose the HDFS block size is 256MB. Suppose to execute a MapReduce application for Hadoop that analyzes the content of *myFolder*. Suppose the map phase emits the following key-value pairs (the key part is a country while the value part is always 1):

    ("Turin", 1)
    ("Turin", 1)
    ("Turin", 1)
    ("Nice", 1)
    ("Paris", 1)

    Suppose the number of instances of the reducer class is set to 4 and suppose the reduce method of the reducer class sums the values associated with each key and emits one pair (country, sum values) for each key. Suppose the following pairs are overall emitted by the reduce phase:

```
(“Turin”, 3)
(“Nice”, 1)
(“Paris”, 1)
```

Considering all the instances of the mapper class, overall, how many times is the ***map method*** invoked?

a) 2

b) 3

c) 4

[✓ d) 5]

2. (2 points) Consider the following Spark application.

```
package it.polito.bigdata.spark.exam;

import ….;

public class SparkDriver {

        public static void main(String[] args) {

                // Create a configuration object and set the name of the application
                SparkConf conf = new SparkConf().setAppName("Spark Code");

                // Create a Spark Context object
                JavaSparkContext sc = new JavaSparkContext(conf);

                // Read input file
                JavaRDD<String> inputRDD = sc.textFile("HumidityReadings.txt");


                // Print on the standard output the total number of input lines
                System.out.println("Total number of lines: " + inputRDD.count());


                // Select the content of the field humidity
                JavaRDD<Double> humiditiesRDD = inputRDD.map(line ->
                                            Double.parseDouble(line.split(",")[1]));
```

```
		// Compute the minimum humidity
		Double minHum = humiditiesRDD.reduce((hum1, hum2) -> {
			if (hum1 < hum2)
				return hum1;
			else
				return hum2;
		});

		// Print on the standard output of minimum humidity
		System.out.println("Min. humidity: " + minHum);



		// Select low humidities
		JavaRDD<Double> lowHumiditiesRDD =
					humiditiesRDD.filter(humidity -> humidity > 40);

		// Store the content of lowHumiditiesRDD
		lowHumiditiesRDD.saveAsTextFile("outputFolder/");


		// Close the Spark context
		sc.close();
	}
}
```

Suppose the input file HumidityReadings.txt is read from HDFS. Suppose you execute this Spark application only 1 time. Which one of the following statements is true?

a) This application reads the content of HumidityReadings.txt 1 time

b) This application reads the content of HumidityReadings.txt 2 times

[✓ c) This application reads the content of HumidityReadings.txt 3 times]

d) This application reads the content of HumidityReadings.txt 6 times

Part II

PoliTV is an international online streaming company focused on movies. Its users can watch the available movies through smart TVs, PCs, or mobile devices. The managers of PoliTV are interested in computing a set of specific statistics based on the following input data sets/files.

- Users.txt
  - Users.txt is a large text file containing the list of registered users of PoliTV. PoliTV has millions of users, i.e., Users.txt has millions of lines.
  - Each line of Users.txt is associated with the profile of one user and has the following format
    - Username,Gender,YearOfBirth,Country

      where, *Username* is the unique user identifier, *gender* is his/her gender, *YearOfBirth* is his/her year of birth, and *Country* is the country where he/she resides.

    - For example, the line

      PaoloG76,Male,1976,Italy

  - means that the user identified by the username *“**PaoloG76**”* is a *“**Male**”, who* was born in ***1976*** and resides in ***Italy***.

- Movies.txt
  - Movies.txt is a large text file containing the catalog of available movies (more than 100000 movies).
  - Each line of Movies.txt is associated with one movie and has the following format
    - MID,Title,Director,ReleaseDate

      where, *MID* is the unique movie identifier, *Title* is its title, *Director* is its director, and *RelaseDate* is the date in which the movie was released.

    - For example, the line

      MID124,Ghostbusters,Ivan Reitman,1984/05/01

      means that the title of movies *“**MID1124**”* is “***Ghostbusters***”, its director is “***Ivan Reitman***” and the movie was released on ***May 1, 1984***.

- WatchedMovies.txt
    - WatchedMovies.txt is a large text file.
    - Every time a user watches a movie, a new line is appended to the end of WatchedMovies.txt. The data collected in the last 10 years are currently stored in WatchedMovies.txt, i.e., WatchedMovies.txt contains billions of lines.
    - Each line of WatchedMovies.txt has the following format
        - Username,MID,StartTimestamp,EndTimestamp

          where *Username* is a user identifier and *MID* is a movie identifier. The meaning of each line is "*Username* watched *MID* from *StartTimestamp* to *EndTimestamp"*.

        - For example, the line

          PaoloG76,MID124,2010/06/01_14:18,2010/06/01_16:10

          means that user ***PaoloG76*** watched movie ***MID124*** from ***14:18 of June 1, 2010*** to ***16:10 of June 1, 2010***.

Exercise 1 – MapReduce and Hadoop (7 points)
The managers of PoliTV are interested in performing some analyses about the watched movies.

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

1. *Movies watched by one single user in year 2019.* The application considers only the visualizations related to year 2019 (i.e., the lines of WatchedMovies with StartTimestamp in the range January 1, 2019 – December 31, 2019) and selects the movies that have been watched by one single user in 2019. Store the identifiers (MIDs) of the selected movies in the output HDFS folder (one MID per line).

   Note. If a movie has been watched many times in 2019 but always by the same user, that movie satisfies the constraint and must be selected.

Suppose that the input is WatchedMovies.txt and it has been already set and also the name of the output folder has been already set.

- Write only the content of the Mapper and Reducer classes (map and reduce methods. setup and cleanup if needed). The content of the Driver must not be reported.

- Use the next two specific multiple-choice questions to specify the number of instances of the reducer class for each job.

- If your application is based on two jobs, specify which methods are associated with the first job and which are associated with the second job.

- If you need personalized classes report for each of them:
  - name of the class
  - attributes/fields of the class (data type and name)
  - personalized methods (if any), e.g, the content of the toString() method if you override it
  - do not report get and set methods. I suppose they are "automatically defined"

Exercise 1 - Number of instances of the reducer - Job 1 - MapReduce and Hadoop (0.5 points)

Select the number of instances of the reducer class of the first Job

(a) 0

(b) exactly 1

[✓ (c) any number >=1]

Exercise 1 - Number of instances of the reducer - Job 2 - MapReduce and Hadoop (0.5 points)

Select the number of instances of the reducer class of the second Job

[✓ (a) One single job is needed for this MapReduce application]

(b) 0

(c) exactly 1

(d) any number >=1

Exercise 2 – Spark and RDDs (19 points)

The managers of PoliTV are interested in performing some analyses about the watched movies.

Develop one single application to address all the analyses they are interested in. The application has five arguments: the three input files Users.txt, Movies.txt, and WatchedMovies.txt and two output folders, “outPart1/” and “outPart2/”, which are associated with the outputs of the following Points 1 and 2, respectively.

Specifically, design a single application, based on Spark, and write the corresponding code, to address the following points:

1. *Movies that have been watched frequently but only in one year in the last five years*. Considering only the lines of WatchedMovies.txt related to the last five years (i.e., the lines with StartTimestamp in the range September 17, 2015 - September 16, 2020), the application selects the movies that (i) have been watched only in one of those 5 years and (ii) at least 1000 times in that year. For each of the selected movies, the application stores in the first HDFS output folder its identifier (MID) and the single year in which it has been watched at least 1000 times (one pair (MID, year) per line).

   Note. The value of StartTimestamp is used to decide in which year a user watched a specific movie. Do not consider the value of EndTimestamp.

   Point 1: Examples

   - For instance, suppose that movie MID15 was watched 1025 times in year 2016 and it was never watched in the other four years. MID15 is selected and the following pair is stored in the output:

     MID15,2016

   - For instance, suppose that movie MID56 was watched 1056 times in year 2016 and 10 times in year 2018. MID56 must not be selected because it has been watched in more than one year (considering the last five years).

2. *Most popular movie in at least two years*. Considering all the lines of WatchedMovies.txt (i.e., all years), the application selects the movies that ***have been*** the most popular movie in at least two years. The annual popularity of a movie in a specific year is given by the number of distinct users who watched that movie in that specific year. A movie is the most popular movie in a specific year if it is associated with the highest annual popularity of that year. The application stores in the second HDFS output folder the identifiers (MIDs) of the selected movies (one MID per line).

   Note. The value of StartTimestamp is used to decide in which year a user watched a specific movie. Do not consider the value of EndTimestamp.

   Point 2: Example

In this toy example, suppose that there are only four movies (MID1, MID2, MID3, and MID4) and only five years (2010, 2011, 2012, 2013, and 2014).

The following table reports, for this toy example, the number of distinct users who watched each of the considered movies in each of the considered years. The most popular movie(s) of each year is (are) highlighted in boldface. Note that many movies can be the most popular movie of the same year (i.e., many movies can be associated with the highest annual popularity of a specific year).

	2010	2011	2012	2013	2014
MID1	**100**	**120**	5	**50**	20
MID2	10	3	15	10	5
MID3	15	10	50	30	**55**
MID4	50	10	**150**	**50**	10

Hence

- MID1 is the most popular movie in years 2010, 2011, and 2013
- MID2 is never the most popular movie
- MID3 is the most popular movie in year and 2014
- MID4 is the most popular movie in years 2012 and 2013

It follows that, movies MID1 and MID4 are selected and stored in the second output folder of this application because they have been the **most popular movie in at least two years**. Movie MID2 and MID3 are discarded because they do not satisfy the constraint.

For the sake of simplicity, in this toy example only four movies and five years are considered but pay attention that WatchedMovies.txt contains the data collected in the last 10 years and the information about the visualizations of more than 100000 movies.

Predefined Template

```
...

/* Suppose all the needed imports are already set */

…

public class SparkDriver {

    public static void main(String[] args) {
        String inUsers;
        String inMovies;
        String inWatchedMovies;
        String outputPathPart1;
        String outputPathPart2;

        inUsers = "Users.txt";
        inMovies = "Movies.txt";
        inWatchedMovies = " WatchedMovies.txt";


        outputPathPart1 = "outPart1/";
        outputPathPart2 = "outPart2/";

        // Create a configuration object and set the name of the application
        SparkConf conf = new SparkConf().setAppName("Spark Exam - Exercise #2");

        // Create a Spark Context object
        JavaSparkContext sc = new JavaSparkContext(conf);

        /* Write your code here */



}
```