Big Data: Architectures and Data Analytics

July 16, 2020

Student ID _____

First Name _____

Last Name _____

The exam lasts **2 hours**

Part I

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

1. (2 points) Consider the input HDFS folder *patchesFolder* that contains the following three files:

   - patches2018.txt: size=513MB
   - patches2019.txt: size=515MB
   - patches2020.txt: size=5MB

   Suppose that you are using a Hadoop cluster that can potentially run up to 10 instances of the mapper class in parallel. Suppose to execute a MapReduce application for Hadoop that selects the lines containing the word “Linux”. This application is based on a Map-only job. The input of this Hadoop application is the HDFS folder *patchesFolder*. The HDFS block size is 512MB. How many instances of the mapper class are created and executed in parallel when this MapReduce application is executed by specifying the folder *patchesFolder* as input?

   a) 10

   b) 5

   c) 4

   d) 3

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);



		JavaRDD<String> temperatureRDD = sc.textFile("temperatures.txt");

		// Select only the lines with temperature > 30°C
		JavaRDD<String> highTempRDD = temperatureRDD.filter(line -> {
			String fields[] = line.split(",");
			float temperature = Float.parseFloat(fields[1]);
			if (temperature > 30)
				return true;
			else
				return false;
		});

		// Print on the standard output of the driver the number of lines with
		// temperature > 30°C

		System.out.println("Number of lines with temp. > 30°C: " +
					highTempRDD.count());



		// Extract the sensor IDs associated with the selected lines
		JavaRDD<String> sidHighTempRDD = highTempRDD.map(line -> {
			String fields[] = line.split(",");
			String sid = fields[0];
			return sid;
		});

		// Count the number of distinct sensor IDs associated with the
		// selected lines and print that number on the standard output of
		// the driver
```

```
                System.out.println("Number of distinct SIDs with temp. > 30°C: "+
                                    sidHighTempRDD.distinct().count());

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

Suppose the input file temperatures.txt is stored in HDFS. Suppose to execute 1 time this Spark application. Which one of the following statements is true?

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

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

c) This application reads 3 times the content of temperatures.txt

d) This application reads 5 times the content of temperatures.txt

Part II

PoliBooks is a large e-commerce company selling books. The management of PoliBooks is interested in analyzing their data to improve the quality of their books. The computed statistics are based on the following input data sets/files.

- Books.txt
  - Books.txt is a large text file containing the catalog of available books. The file contains one single line for each managed book.
  - Each line of Books.txt is related to one book and has the following format
    - BID,title,genre,publisher,yearOfpublication

      where, *BID* is the book identifier, *title* is its title, *genre* is the genre (e.g., Adventure, Romance, Mystery) of the book, *publisher* is the name of its publisher, and *yearOfpublication* is the year when it has been published. Each book is associated with one single genre.

    - For example, the line

      B1020,The Body in the Library,Crime,Dodd and Company,1942

      means that the title of book ***B1020*** is "***The Body in the Library***", its genre is "***Crime",*** its publisher is "***Dodd and Company***", and it has been published in 1942.

- Purchases.txt
  - Purchases.txt is a text file containing the list of purchases of the last 15 years. Every time a customer buys a book a new line is appended at the end of Purchases.txt.
  - Each line of Purchases.txt is related to one purchase and has the following format
    - Customerid,BID,date,price

      where *Customerid* is a customer identifier and *BID* is the identifier of the book that *Customerid* bought on date *date*. The cost of the purchase is equal to *price*.

    - For example, the line

      customer1,BID20,20170502,19.99

      means that customer ***customer1*** bought the book ***BID20*** on ***May 2, 2017*** and the price of the purchase was ***19.99***€

Exercise 1 – MapReduce and Hadoop (7 points)

The managers of PoliBooks are interested in performing some analyses about the purchased books.

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

1. *Customers who bought the same book at least two times in year 2018.* The application considers only the purchases of year 2018 and selects the identifiers of the customers who bought the same book at least two times in year 2018. For each of the selected customers store in the output HDFS folder one line for each of the books he/she bought at least two times in year 2018. Each output line has the format Customerid\tBID.

   Example

   Suppose that during year 2018 customer **customer1** bought (i) 3 times book **BID25** (ii) 2 times book **BID30**, and (iii) 1 time book **BID50.** Two lines are stored for customer **customer1** in the output folder. Specifically, the following two lines are stored in the output folder:

   customer1\t**BID25**
   customer1\t**BID30**

Suppose that the input is Purchases.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 PoliBooks are interested in performing some analyses about the purchased books.

Develop one single application to address all the analyses they are interested in. The application has four arguments: the two input files Books.txt and Purchases.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. *Maximum number of daily purchases per book in year 2018*. Consider only the purchases of years 2018 and only the books with at least one purchase in year 2018. The application computes the **maximum** number of daily purchases for each book. The application stores in the first HDFS output folder for each book its identifier and its maximum number of daily purchases (one pair (BID, maximum number of daily purchases) per line).

   Point 1: Example

   Suppose for instance that the following lines are the ones related to the purchases in year 2018 of the book identified by BID10.

   customer1,BID10,20180502,19.99
   customer2,BID10,20180502,19.99
   customer3,BID10,20180502,19.99
   customer3,BID10,20180612,18.99
   customer4,BID10,20180612,18.99

   Based on these data, BID10 was bought 3 times in May 2, 2018 and 2 times in June 12, 2018. It follows that the maximum number of daily purchases for book BID10 is 3. The pair (BID10,3) will be stored in one output line of the first output folder.

2. *Windows of three consecutive days with many purchases*. Consider only the purchases of years 2018. The application must select, for each book, all the windows of three consecutive dates such that each date of the window is characterized by a number of purchases that is greater than 10% of the purchases of the considered book in year 2018. Specifically, given a book and a window of three consecutive dates, that window of three consecutive dates is selected for that book if and only if in each of those three dates the number of purchases of that book is greater than 0.1*total number of purchases of that book in year 2018. The application stores the result in the second HDFS output folder. Specifically, each of the selected combinations (book, window of three consecutive dates) is stored in one output line and the used format is the following: (BID of the selected book, first date of the selected window of three consecutive dates).

Note that you can have overlapped windows of three consecutive dates among the selected windows.

Suppose that someone has already implemented the following Java function

- *public static String previousDeltaDate(String inputDate, int deltaDays)*
    - The parameters of this function are
        - *inputDate*: this is a string representing a date (in the format yyyymmdd)
        - *deltaDays*: this is an integer representing a delta in terms of number of days (e.g., 2 means a delta of two days).
    - The returned value is a string representing the date *inputDate-deltaDays*
    - For example, the invocation

      yesterday= DateTool.previousDeltaDate(“20150502”,1)

      returns the string “20150501” and stores “20150501” in the variable *yesterday.*

Point 2: Examples

For instance, suppose that book BID10 was purchases 100 times in year 2018 and suppose that BID10 was purchased 20 times in May 2, 2018, 11 times in May 3, 2018, and 12 times in May 4, 2018. It means that this window of three consecutive dates **must be selected** for BID10 and the following line must be stored in the output folder: (BID10, 20150502).

For instance, suppose that book BID12 was purchases 1000 times in year 2018 and suppose that BID12 was purchased 200 times in June 2, 2018, 35 times in June 3, 2018, and 300 times in June 4, 2018. It means that this window of three consecutive dates **must not be selected** for BID12.

Predefined Template

...

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

…

```
public class SparkDriver {

    public static void main(String[] args) {
        String inputPathServers;
        String inputPathPatchesServers;
        String outputPathPart1;
        String outputPathPart2;

        inputPathbooks = "Books.txt";
        inputPathPurchases = "Purchases.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 */



}
```