Big Data: Architectures and Data Analytics

July 2, 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 *patchesFolder* that contains the following two files:

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

   Suppose that you are using a Hadoop cluster that can potentially run up to 5 instances of the mapper class in parallel. Suppose to execute a MapReduce application for Hadoop that computes some statistics about the patches. This application is based on one single MapReduce job. The input of this Hadoop application is the HDFS folder *patchesFolder*. The HDFS block size is 512MB. The number of instances of the reducer class is set to 2. How many mappers are instantiated by Hadoop (i.e., how many instances of the mapper class) when you execute this MapReduce application by specifying the folder *patchesFolder* as input?

   a) 5

   b) 4

   c) 3

   d) 2

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 Accumulator");

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

		// Define an accumulator of type long
		final LongAccumulator invalidEmails = sc.sc().longAccumulator();


		// Create an RDD of Strings from a local list.
		List<String> inputList= Arrays.asList(
				"paolo.garza@polito.it",
				"paolo.garza@polito.it",
				"p.garza_polito.it",
				"test_polito.it");

		JavaRDD<String> emailsRDD = sc.parallelize(inputList);

		// Select only valid emails, i.e., emails that contain the @ symbol.
		JavaRDD<String> validEmailsRDD = emailsRDD.filter(line ->
		{
			// Increase the accumulator if the line contains an invalid email
			if (line.contains("@")==false)
				invalidEmails.add(1);
			return line.contains("@");
		});

		// Print on the standard output of the Driver
		// - Number of valid emails (by considering also duplicates)
		// - Number of distinct valid emails
		// - Number of invalid emails

		long numValidEmails =  validEmailsRDD.count();
		System.out.println("Number of valid emails (by considering also
				duplicates): "+ numValidEmails);

		long numDistinctValidEmails =  validEmailsRDD.distinct().count();
		System.out.println("Number of distinct valid emails: " +
				numDistinctValidEmails);

		long numInvalidEmails =  invalidEmails.value();
		System.out.println("Number of invalid emails: "+ numInvalidEmails);
```

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

Which one of the following statements is true?

a) The execution of this application prints on the standard output of the driver the following result

   Number of valid emails (by considering also duplicates): 2
   Number of distinct valid emails: 1
   Number of invalid emails: 4

b) The execution of this application prints on the standard output of the driver the following result

   Number of valid emails (by considering also duplicates): 2
   Number of distinct valid emails: 1
   Number of invalid emails: 2

c) The execution of this application prints on the standard output of the driver the following result

   Number of valid emails (by considering also duplicates): 2
   Number of distinct valid emails: 1
   Number of invalid emails: 0

d) Accumulators cannot be incremented inside an RDD filter transformation.

Part II

PoliCompany is an international company with several data centers around the world. The managers of PoliCompany are interested in monitoring the patches applied on the servers of the managed data centers. The computed statistics are based on the following input data sets/files.

- Servers.txt
  - Servers.txt is a text file containing the information about the servers of PolyCompany. PoliCompany has more than 50000 servers.
  - Each line of Servers.txt is related to one server and has the following format
    - SID,Model

      where *SID* is the server identifier and *Model* is its model.

    - For example, the following line

S10,SunP20

means that the model of the server with id ***S10*** is ***SunP20***.

- PatchedServers.txt
  - PatchedServers.txt is a text file containing the information about which patches had been applied on each server in the last 10 years (2010-2019). Every time a patch is applied on a server, a new line is inserted in PatchedServers.txt.
  - Each line of PatchedServers.txt has the following format
    - SID,PID,Date

      where *SID* is the identifier of the server on which the patch *PID* was applied on the date *Date.*

    - For example, the following line

      S10,P12,2019/08/05

      means that the patch with id ***P12*** was applied on server ***S10*** on ***August 5, 2019***.

Exercise 1 – MapReduce and Hadoop (7 points)

The managers of PoliCompany are interested in performing some analyses about their servers and the applied patches.

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

1. *Servers with many patches in 2017 or 2018*. The application considers only the data of years 2017 and 2018 and selects the identifiers (SIDs) of the servers on which at least 30 patches were applied in at least one of the two considered years (i.e., at least 30 patches in 2017 or at least 30 patches in 2018). Store the identifiers (SIDs) of the selected servers in the output HDFS folder (one selected SID per line).

   Examples

   Suppose 35 patches were applied in year 2017 on server S10 and 1 in year 2018. Server S10 **is selected** and stored in the output folder.

   Suppose 15 patches were applied in year 2017 on server S20 and 18 in year 2018. Server S20 **is not selected** because there are less than 30 patches in both years.

   Suppose 32 patches were applied in year 2017 on server S30 and 31 in year 2018. Server S30 **is selected** and stored in the output folder.

Suppose that the input is PatchedServers.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 PoliCompany are interested in performing some analyses about their servers and the applied patches.

The managers of PoliCompany 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 Servers.txt and PatchedServers.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. *Servers with a high decrease of the number of applied patches in the least two years.* The application selects the servers with a high decrease of the number of patches from year 2018 to year 2019. Specifically, a server is selected if the number of patches applied in year 2019 on that server is less than 50% of the number of patches applied in year 2018 on the same server (i.e., a server is selected if for that server num. applied patches in 2019 < 0.5 * num. applied patches in 2018). The application stores the identifiers and the models of the selected servers in the first HDFS output folder. Each output line contains the **SID and** the **model** of one of the selected servers (one SID,Model per line).

   Point 1: Examples

   - Suppose that 10 patches were applied on server S10 in year 2018 and 4 patches in year 2019. **Server S10 is selected** because the number of patches applied on that server in year 2019 is less than 50% of the ones applied in year 2018 on the same server (4 < 0.5*10). The output line associated with server S10 contains its identifier and its model (i.e., S10, SunP20).

   - Suppose that 20 patches were applied on server S20 in year 2018 and 12 patches in year 2019. **Server S20 is not selected** because the number of patches applied on that server in year 2019 is greater than 50% of the ones applied in year 2018 on the same server (12 > 0.5*20).

   - Suppose that 5 patches were applied on server S30 in year 2018 and 0 patches in year 2019. **Server S30 is selected** because the number of patches applied on that server in year 2019 is less than 50% of the ones applied in year 2018 on the same server (0 < 0.5*5). The output line associated with server S30 contains its identifier and its model (i.e., S30, HPCompact80).

2. *Servers with at most one applied patch per date*. The application must select the servers on which at most one patch in each date has been applied (i.e., servers with 0 or 1 patch in each date). The application **stores the selected servers in the second HDFS output folder**. Each output line contains **the SID** and the **model** of one of the selected servers (one pair SID,Model per line). The application also **prints on the standard output of the driver the number of distinct models** associated with the selected servers.

   Point 2: Example

   Suppose the content of Servers.txt is the following (this is a small running example that is used to clarify the problem specification. The real file contains more than 50000 lines)

   S10,SunP20
   S11,IBM125
   S12,IBM80
   S13,SunP20

   Suppose the content of PatchedServers.txt is the following (this is a small running example that is used to clarify the problem specification. The real file contains more than 5000000 lines)

   S10,P12,2018/01/10
   S10,P13,2019/08/09
   S10,P14,2019/08/10
   S12,P12,2019/08/07
   S12,P13,2019/08/07
   S12,P15,2019/08/10
   S13,P15,2019/12/01

   With this running example data, the result of Point 2 will be the following:

   - Content of the second HDFS output folder

     S10,SunP20
     S11,IBM125
     S13,SunP20

   - Number of distinct models of the selected servers that is print on the standard output of the driver

     Number of distinct models of the selected servers: 2

   Pay attention that there are servers without applied patches (i.e., some servers occur in Servers.txt but not in PatchedServers.txt). For instance, in the running example server S11 occurs only in Servers.txt.

- 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"

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;

        inputPathServers = "Servers.txt";
        inputPathPatchesServers = "PatchedServers.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 */



}
```