Distributed architectures for big data processing and analytics

September 17, 2021

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 following Spark application.

```
inputRDD = sc.textFile("HumidityValues.txt")
# Compute the total number of lines
totLines=inputRDD.count()

# Print on the standard output the total number of input lines
print("Total number of lines: " + str(totLines) + "\n")

# Select the content of the field humidity
humiditiesRDD = inputRDD.map(lambda line: float(line.split(",")[1]))

# Compute the minimum humidity
minHum = humiditiesRDD.reduce(lambda hum1, hum2: min(hum1, hum2))
# Print on the standard output the minimum humidity
print("Min. humidity: " + str(minHum) +"\n")


#Compute the maximum humidity
maxHum = humiditiesRDD.reduce(lambda hum1, hum2: max(hum1, hum2))
# Print on the standard output the maximum humidity
print("Max. humidity: " + str(maxHum) +"\n")

# Print on the standard output maximum humidity- minimum humidity
print("Max. humidity: " + str(maxHum-minHum) +"\n")
```

Suppose the input file HumidityValues.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 HumidityValues.txt 2 times

b) This application reads the content of HumidityValues.txt 3 times

c) This application reads the content of HumidityValues.txt 4 times

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

2. (2 points) Consider the HDFS file data.txt. The size of data.txt is 10000MB. Suppose that you are using a Hadoop cluster that can potentially run up to 20 instances of the mapper class in parallel and suppose to execute a MapReduce-based program that selects the rows of data.txt containing the words “BIG” and “DATA”. Which of the following values is a proper HDFS block size if you want to “force” Hadoop to run exactly 5 instances of the mapper class in parallel when you execute the application by specifying data.txt as input file?

a) Block size: 2048MB

b) Block size: 1024MB

c) Block size: 512MB

d) Block size: 256MB

Part II

PoliOnline is an international company that sells items online. To improve the number of sales and revenue of PoliOnline, a set of statistics about the managed items and customers are computed based on the following input data sets/files.

- ItemsCatalog.txt
  - ItemsCatalog.txt is a textual file containing the information about the items that are sold by PoliOnline. There is one line for each item and the total number of items is greater than 1,000,000. This file is large and you cannot suppose the content of ItemsCatalog.txt can be stored in one in-memory python variable.
  - Each line of ItemsCatalog.txt has the following format
    - ItemID,Name,Category,FirstTimeInCatalog

      where *ItemID* is the item unique identifier, *Name* is the name of *ItemID*, *Category* is its category (i.e., the item category), and *FirstTimeInCatalog* is the first timestamp in which *ItemID* was included in the catalog of PoliOnline. The format of FirstTimeInCatalog is "YYYY/MM/DD-HH:MM:SS".

    - For example, the following line

      ID1,t-shirt-winter,Clothing,2001/03/01-12:00:00

      means that the item with ItemID ***ID1*** is characterized by the name ***t-shirt-winter***, it belongs to the ***Clothing*** category, and it was included in the PoliOnline's catalog on March 1, 2001 at 12:00:00.

- Customers.txt
  - Customers.txt is a textual file containing the information about the customers who are registered on the web site of PoliOnline. There is one line for each customer and the total number of customers is greater than 10,000,000. This file is large and you cannot suppose the content of Customers.txt can be stored in one in-memory python variable.
  - Each line of Customers.txt has the following format
    - Username,Name,Surname,DateOfBirth

      where *Username* is the customer unique identifier, *Name* and *Surname* are his/her name and surname, respectively, and DateOfBirth is his/her date of birth. The DateOfBirth format is "YYYY/MM/DD".

    - For example, the following line

      User20,Paolo,Garza,1976/03/01

means that the name and surname of customer ***User20*** are ***Paolo*** and ***Garza***, respectively, and that the customer was born on March 1, 1976.

- Purchases.txt
  - Purchases.txt is a textual file containing information about purchases/sales. A new line is inserted in Purchases.txt every time an item is bought by a customer. Purchases.txt contains the historical data about the last 10 years. This file is big and you cannot suppose the content of Purchases.txt can be stored in one in-memory python variable.
  - Each line of Purchases.txt has the following format
    - SaleTimestamp,Username,ItemID,SalePrice

      where *SaleTimestamp* is the timestamp at which the customer *Username* bought the item identified by *ItemID*. *SalePrice* is the price at which *Username* bought *ItemID.*

    - For example, the following line

      2019/02/02-09:15:01,User20,ID1,50.99

      means that on ***February 2, 2019***, at ***09:15:01*** the item identified by ***ID1*** was bought by customer ***User20***, and ***User20*** bought that item for ***50.99*** euro. The format of SaleTimestamp is "YYYY/MM/DD-HH:MM:SS".

      Note that the same customer can buy the same item multiple times, in different timestamps.

Exercise 1 – MapReduce and Hadoop (7 points)

The managers of PoliOnline are interested in performing some analyses about the effectiveness of their advertisements.

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

1. *Customer with the maximum number of purchases in the year 2010 and no purchases in all the other years.* The application considers all the purchases (each line of Purchases.txt corresponds to one purchase) and selects the customer who satisfies the following two constraints: (i) the selected customer is the one with the maximum number of purchases in the year 2010 and (ii) the selected customer never purchased products in all the other years. In case of a tie, among the customers with the maximum number of purchases in the year 2010 and no purchases in the other years, the customer associated with the first Username (according to the alphabetic order) is selected. Store the identifier (Username) of the selected customer and the number of purchases in 2010 of that customer in the output HDFS folder (Username\tNumber of purchases in 2010 of Username).

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 (19 points)

The managers of PoliOnline are interested in performing some analyses related to their data.

The managers of PoliOnline asked you to develop one single application to address all the analyses they are interested in. The application has five arguments: the three input files ItemsCatalog.txt, Customers.txt, 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 RDDs or Spark DataFrames, and write the corresponding Python code, to address the following points:

1. *Items that were purchased at least 1000 times in each year from 2010 to 2019.* Considering only the purchases related to the period 2010-2019 (i.e., the 10 years from 2010 to 2019), the application selects the items that are frequently purchased in each of the 10 years of the considered period. An item is a frequently purchased item if it was purchased at least 1000 times in each of the 10 years we are considering. The application stores in the first HDFS output folder the identifiers (ItemIDs) of the selected frequently purchased items (one ItemID per output line).

2. *Number of items that have never been sold in the two years after their inclusion in the PoliOnline's catalog for each category.* The application, for this second analysis, considers all the purchases in Purchases.txt related to the items that were included in the catalog of PoliOnline before January 1, 2010 (FirstTimeInCatalog before January 1, 2010). The application selects the items that have never been sold in the two years after their inclusion in the PoliOnline's catalog (these items are referred to as NotSoldFirst2Years in the following). The application computes for each category the number of NotSoldFirst2Years items and selects only the categories with at least 50 NotSoldFirst2Years items. The selected categories are stored in the second output folder. Each line of the output folder contains one of the selected categories and the number of NotSoldFirst2Years items of that category.

   Suppose that someone already defined the python function *computeDifference(timestamp1, timestamp2)*. Given two timestamps, *computeDifference(timestamp1, timestamp2)* returns the difference between *timestamp1* and *timestamp2* in years (the return value is a float). *timestamp1* and *timestamp2* use the same format of *SaleTimestamp* and *FirstTimeInCatalog*.

   Suppose sc (Spark Context) and spark (Spark Session) have been already set.

```
itemsPath= 'ItemsCatalog.txt'
customersPath= 'Customers.txt'
purchasesPath= 'Purchases.txt'

output1 = 'out1'
output2 = 'out2'
```