Distributed architectures for big data processing and analytics

February 10, 2022

Student ID _____

First Name _____

Last Name _____

The exam lasts **90 minutes**

Part I

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

1. (2 points) Consider the following Spark Streaming applications.

```
from pyspark.streaming import StreamingContext
# Create a Spark Streaming Context object
ssc = StreamingContext(sc, 10)

# Create a (Receiver) DStream that will connect to localhost:9999
inputDStream = ssc.socketTextStream("localhost", 9999)

# Part A
# Define windows and map input strings to integers
inputAWindowDStream = inputDStream\
.window(20, 10)\
.map(lambda value: int(value))

#Apply a filter
filteredADStream = inputAWindowDStream.filter(lambda value : value>5)

# Compute the maximum value
resADStream = filteredADStream.reduce(lambda v1,v2:max( v1,v2))

# Print the result on standard output
resADStream.pprint()

# Part B
# Map input strings to integers
inputBDStream = inputDStream\
.map(lambda value: int(value))

#Apply a filter, compute max, and define windows
filteredBDStream = inputBDStream.filter(lambda value : value>5)\
.reduce(lambda v1,v2:max( v1,v2))
.window(20, 10)

# Compute the maximum value again
resBDStream = filteredBDStream.reduce(lambda v1,v2:max( v1,v2))

# Print the result
resBDStream.pprint()
```

```
# Part C
# Map input strings to integers and define windows
inputCWindowDStream = ssc.socketTextStream("localhost", 9999)\
.map(lambda value: int(value))\
.window(20, 10)

# Compute the maximum value
maxCWindowDStream = inputCWindowDStream.reduce(lambda v1,v2:max( v1,v2))

#Apply a filter
resCDStream = maxCWindowDStream.filter(lambda value : value>5)

# Print the result
resCDStream.pprint()

ssc.start()
ssc.awaitTerminationOrTimeout(360)
ssc.stop(stopSparkContext=False)
```

Which one of the following statements is true?

a) Independently of the content of **inputDStream**, **resADStream**, **resBDStream**, and **resCDStream** contain always the same integer values.
b) Independently of the content of **inputDStream**, **resADStream** and **resBDStream** contain always the same integer values, while **resCDStream** may contain different integer values with respect to **resADStream** and **resBDStream**.
c) Independently of the content of **inputDStream**, **resADStream** and **resCDStream** contain always the same integer values, while **resBDStream** may contain different integer values with respect to **resADStream** and **resCDStream**.
d) Independently of the content of **inputDStream**, **resBDStream** and **resCDStream** contain always the same integer values, while **resADStream** may contain different integer values with respect to **resBDStream** and **resCDStream**.

2. (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")
```

```
# Select the occurrences of the minimum humidity
minOccurrrences = humiditiesRDD.filter(lambda value: value== minHum)

# Compute the number of occurrences of the minimum humidity
numOccs = minOccurrrences.count()

# Print on the standard output the number of occurrences of the minimum humidity
print("Num occurrences min. humidity: " + str(numOccs) +"\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 1 time
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

Part II

PoliMarketplace is a marketplace that is used to sell mobile apps. PoliMarketplace manages millions of apps and is used by millions of users. PoliMarketplace computes statistics about the usage of its apps and the characteristics of its users. The analyses are based on the following input data sets/files.

- Apps.txt
  - Apps.txt is a text file containing the list of mobile apps available on PoliMarketplace. Each line of Apps.txt is associated with one app. PoliMarketplace manages millions of apps.
  - Each line of Apps.txt has the following format
    - AppId,AppName,Price,Category,Company

      where *AppId* is the unique identifier of the app while *AppName* is its name, *Price* is its price, *Category* is its category (game, office, finance, etc.), and *Company* is the company that developed the app.

    - For example, the following line

      App10,PolitoApp,0,Education,Polito

      means that the app identified by the AppId ***App10*** is called PolitoApp, it costs ***0*** euros, it belongs to the ***Education*** category, and it is developed by ***Polito***.

- Users.txt
  - Users.txt is a text file containing the profiles of the users of PoliMarketplace. Each line is associated with one user. PoliMarketplace has millions of users.
  - Each line of Users.txt has the following format
    - UserId,Name,Surname,Nationality

      where *UserId* is the unique identifier of the user while *Name* and *Surname* are his/her name and surname, respectively. Nationality is his/her nationality.

- For example, the following line

  User15,Paolo,Garza,Italian

  means that the name and surname of the user identified by the id ***User15*** are ***Paolo*** and ***Garza***, respectively, and User15 is ***Italian***.

- Actions.txt
  - Actions.txt is a text file that is used to track installations and removals of apps. A new line is appended at the end of Actions.txt every time a user installs or removes an app. Actions.txt stores more than 20 years of data.
  - Each line of Actions.txt has the following format
    - UserId,AppId,Timestamp,Action

      where *UserID* is the identifier of the user who performed the action specified in the field *Action* on the app with id *AppId* at time *Timestamp*. *Action* can assume one of the following three values: "Install" or "Remove". Pay attention that the same user can perform the same action on the same app multiple times, in different timestamps (i.e., a user can install or remove the same app multiple times). The field *Timestamp* is a string and its format is "YYYY/MM/DD-HH:MM:SS".

    - For example, the following line

      User15,App10,2019/01/01-23:01:15,Install

      means that ***User15*** installed (Action is equal to ***Install***) the app ***App10*** on ***January 1, 2019,*** at ***23:01:15***.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliMarketplace are interested in performing some analyses about their apps.

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

1. *Companies with only free apps considering the category Education.* The application considers only the apps of the category Education and selects the companies that developed only free apps (considering only the category Education). An app is free if its price is equal to 0. Store the selected companies and the number of apps of the category Education developed by each of the selected companies in the output HDFS folder (one pair (company, number of developed apps of the category Education) per output line).

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

- **Write your code on your papers.**

- 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 following two specific multiple-choice questions (**Exercises 1.2 and 1.3)** 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:
  - the 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 the get and set methods. Suppose they are "automatically defined"

Answer the following two questions to specify the number of jobs (one or two) and the number of instances of the reducer classes.

Exercise 1.2 - Number of instances of the reducer - Job 1

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

- (a) 0
- (b) exactly 1
- (c) any number >=1 (i.e., the reduce phase can be parallelized)

Exercise 1.3 - Number of instances of the reducer - Job 2

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

- (a) One single job is needed
- (b) 0
- (c) exactly 1
- (d) any number >=1 (i.e., the reduce phase can be parallelized)

Exercise 2 – Spark and RDDs (19 points)

The managers of PoliMarketplace asked you to develop one single application to address all the analyses they are interested in. The application has five arguments: the input files Apps.txt, Users.txt, and Actions.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. *Users who installed a number of non-free apps greater than the number of free apps in the year 2022.* This first part of the application considers only the data related to the year 2022 and selects the users who installed a number of distinct non-free apps greater than the number of distinct installed free apps. A user installed a specific app in the year 2022 if there is at least one line in Actions.txt (considering only the lines related to the year 2022) associated with that pair (user, app) for which Action is equal to ‘Install’. For each of the selected users, store in the first HDFS output folder (one user per line) his/her userId and the number of distinct installed non-free apps in the year 2022.

2. *Italian users with the maximum number of currently installed apps.* This second part of the application considers only Italian users and selects the Italian users with the maximum number of currently installed apps. An app is currently installed on the smartphone of a user if the last action executed by that user associated with that app is the action 'Install' (Action='Install'). The UserIds of the selected Italian users are stored in the second HDFS output folder (one UserId per output line).
   Pay attention. All the users associated with the maximum number of currently installed apps are stored in the output folder (one UserId per output line).

   Examples Point 2

   - *First example.* For the sake of clarity, suppose that there are only three Italian users: User1, User 2, and User 3. Suppose that (i) the number of currently installed apps for User1 is 10, (ii) the number of currently installed apps for User2 is 35, and the number of currently installed apps for User3 is 29. In this first example, the userId **User2** is stored in the second output folder.
   - *Second example.* For the sake of clarity, suppose that there are only three Italian users: User1, User 2, and User 3. Suppose that (i) the number of currently installed apps for User1 is 43, (ii) the number of currently installed apps for User2 is 10, and the number of currently installed apps for User3 is 43. In this second example, the userIds **User2** and **User3** are stored in the second output folder.
   - Pay attention that the actual input file contains millions of users.

   - **Write your code on your papers.**
   - You do not need to report imports. Focus on the content of the main method.
   - Suppose both **JavaSparkContext sc** and **SparkSession ss** have been already set.
   - Suppose the following variables have been already set:
     - usersPath= 'Users.txt'
       appsPath= 'Apps.txt'
       actionPath= 'Actions.txt'
       output1 = outPart1/'
       output2 = outPart2/'
   - If you need personalized classes, report for each of them:
     - the 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 the get and set methods. Suppose they are "automatically defined"