Distributed architectures for big data processing and analytics

September 18, 2023

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 application.

```
# RDDs associated with the input files
inputRDD1 = sc.textFile("TempReadings1.txt")
inputRDD2 = sc.textFile("TempReadings2.txt")

# Union inputRDD1 and inputRDD2

inputRDD = inputRDD1.union(inputRDD2)

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

# Select high temperatures
highTempsRDD = tempsRDD.filter(lambda temp: temp>40)

# Store the content of highTempsRDD
highTempsRDD.saveAsTextFile("outputFolderHigh/")

# Select low temperatures
lowTempsRDD = tempsRDD.filter(lambda temp: temp<-20)

# Store the content of lowTempsRDD
lowTempsRDD.saveAsTextFile("outputFolderLow/")
```

Suppose the input files TempReadings1.txt and TempReadings2.txt are 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 TempReadings1.txt 0 times and the content of TempReadings2.txt 0 times
b) This application reads the content of TempReadings1.txt 1 time and the content of TempReadings2.txt 1 time
c) This application reads the content of TempReadings1.txt 2 times and the content of TempReadings2.txt 2 times
d) This application reads the content of TempReadings1.txt 3 times and the content of TempReadings2.txt 3 times

2. (2 points) Consider the input HDFS folder myFolder. myFolder contains the following two files:

- ProfilesItaly.txt
  - The text file ProfilesItaly.txt contains the following five lines:
    Luca,Rome
    Luca,Rome
    Carmen,Naples
    Luca,Turin
    Carmen,Milan

- ProfilesFrance.txt
  - The text file ProfilesFrance.txt contains the following three lines:
    Danilo,Paris
    Carmen,Paris
    Pablo,Nice

Suppose you are using a Hadoop cluster that can run up to 5 instances of the mapper class in parallel. Suppose the HDFS block size is 512MB. Suppose to execute a MapReduce application for Hadoop that analyzes the content of myFolder. Suppose the map phase emits, overall, the following key-value pairs (the key part is a name while the value part is the length of the name):

(Luca, 4)
(Luca, 4)
(Carmen, 6)
(Luca,4)
(Carmen,6)
(Danilo,6)
(Carmen,6)
(Pablo,5)

Suppose the number of instances of the reducer class is set to 3. Suppose the reduce method of the reducer class sums the values associated with each key and emits one pair (name, sum values) for each key for which the sum is greater than 10. Specifically, suppose the following pairs are overall emitted by the reduce phase:

(Luca, 12)
(Carmen, 18)

Considering all the reducer class instances, how many times is the **reduce method** invoked?

a) 2
b) 3
c) 4
d) 8

Part II

OnlineTVSeries is an international company operating worldwide. OnlineTVSeries is specialized in streaming television series on demand. It manages several television series and has millions of customers. Statistics about the television series and the customers are computed based on the following input data files, which have been collected in the company's latest twenty years of activity.

- Customers.txt
    - Customers.txt is a textual file containing information about the customers of OnlineTVSeries. There is one line for each customer. The total number of customers is greater than 300,000,000. Customers.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
    - Each line of Customers.txt has the following format
        - <u>CID</u>,Name,Surname,City,Country

          where *CID* is the customer's unique identifier, *Name* and *Surname* are his/her name and surname, respectively, and *City* and *Country* are the city and country where he/she lives, respectively.
        - For example, the following line

          CID10,John,Bianco,Turin,Italy

          means that the name and surname of the customer with identifier ***CID10*** are ***John*** and ***Bianco***, respectively, and he lives in ***Turin*** (***Italy***).

- TVSeries.txt
    - TVSeries.txt is a textual file containing information about the television series (TV series) streamed on OnlineTVSeries. There is one line for each TV series. The total number of television series stored in TVSeries.txt is greater than 100,000. TVSeries.txt is large. Its content cannot be stored in one in-memory Java/Python variable.

- Each line of TVSeries.txt has the following format
  - <u>SID</u>,Title,Genre

    where *SID* is the TV series's unique identifier, *Title* is its title, and *Genre* is its genre.
  - For example, the following line

    SID15,Friends,Comedy

    means that the TV series with SID ***SID15*** is titled ***Friends*** and is a ***Comedy*** television series.

Note that each television series is associated with one single genre.

- Episodes.txt
  - Episodes.txt is a textual file containing information about the episodes of the television series. There is one line for each episode. The total number of episodes stored in Episodes.txt is greater than 3,000,000. Episodes.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
  - Each line of Episodes.txt has the following format
    - <u>SID,SeasonNumber,EpisodeNumber</u>,Title, OriginalAirDate

      where *SID* is the identifier of the TV series the episode belongs to and *SeasonNumber* is the number of the season this episode is part of. *EpisodeNumber* is the number of this episode in the season number *SeasonNumber* of the TV series identified by *SID*. Each episode is uniquely identified by the triplet (*SID, SeasonNumber, EpisodeNumber*), i.e., the triplet (*SID, SeasonNumber, EpisodeNumber*) is the "primary key" of this file. Finally, *Title* is the episode's title, while *OriginalAirDate* is the date on which the episode was aired/broadcast for the first time.

      OriginalAirDate is a date in the format YYYY/MM/DD.
    - For example, the following line

      SID15,2,7,The One with the Blackout,1994/11/03

      means that the ***7th*** episode of the ***2nd*** season of the television series with SID ***SID15*** is titled **"*The One with the Blackout*"** and was aired/broadcast for the first time on ***November 3, 1994***.

- CustomerWatched.txt
  - CustomerWatched.txt is a textual file containing information about who watched which episodes. A new line is inserted in CustomerWatched.txt every time a customer watches an episode. CustomerWatched.txt contains the historical data about the last 20 years. CustomerWatched.txt is large and cannot be stored in one in-memory Java/Python variable.
  - Each line of CustomerWatched.txt has the following format
    - <u>CID,StartTimestamp</u>,SID,SeasonNumber,EpisodeNumber

      where *CID* is the identifier of the customer who started watching the episode identified by the triplet (*SID*, *SeasonNumber*, *EpisodeNumber*) at the time *StartTimestamp*. *StartTimestamp* is a timestamp in the format YYYY/MM/DD-HH:MM.

- For example, the following line
  CID10,2022/11/07-21:40,SID15,1,7

  means that at ***21:40*** on ***November 7, 2022***, the customer with id ***CID10*** started watching the episode identified by the triplet (***SID15,1,7****)*.

Note that each customer can watch many episodes in different timestamps, and each episode can be watched by many customers. Moreover, **the same customer can watch each episode several times** (a new line associated with a different StartTimestamp is inserted in CustomerWatched.txt for each visualization of the same episode by the same customer). Note that each pair (CID, StartTimestamp) occurs at most one time in CustomerWatched.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of OnlineTVSeries are interested in performing some statistics on the television series.

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

1. *Longest life-spanning television series*. This application computes the lifespan for each television series and selects the TV series associated with the longest lifespan value. The lifespan of a television series is defined as the difference between the first and the latest on-air dates associated with its episodes. If many TV series are associated with the longest lifespan value, select the one with the first SID according to the alphabetical order. Store the SID of the selected television series in the HDFS output folder.

Suppose there is a function called Diff.diffDatesInDays(firstDate,secondDate) that returns the difference between secondDate and firstDate in terms of number of days. For instance, the invocation *Diff.diffDatesInDays*(“*2022/11/07*”, “*2022/11/13”*) returns *6*.

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

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 (19 points)

The managers of OnlineTVSeries asked you to develop one single application to address all the analyses they are interested in. The application has six arguments: the input files Customers.txt, TVSeries.txt, Episodes.txt, and CustomerWatched.txt, and the two output folders “outPart1/” and “outPart2/”, which are associated with the outputs of 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 two points:

1. *Total number of episodes for each comedy television series with a lifespan of at least 3650 days*. The first part of this Spark application considers only the comedy television series (genre equal to comedy) with a lifespan of at least 3650 days. The lifespan of a television series is defined as the difference between the first and the latest on-air dates associated with its episodes. For each comedy TV series with a lifespan of at least 3650 days, compute the total number of episodes. Store the result in the first HDFS output folder. The output contains one line for each comedy TV series with a lifespan of at least 3650 days. Each output line is formatted as follows:
   SID,Total number of episodes for this SID

   Suppose there is a function called *diffDatesInDays(firstDate,secondDate)* that returns the difference between secondDate and firstDate in terms of number of days. For instance, the invocation *diffDatesInDays*(“*2022/11/07*”, “*2022/11/13”*) returns *6*.

2. *The number of TV series completely watched by each customer*. The second part of the Spark application considers all television series and computes, for each customer,

the number of TV series for which the customer watched all episodes. Store the result in the second HDFS output folder. Specifically, the second output folder must contain one line for each customer with the following information:

CID,Number of TV series for which the customer CID watched all episodes

Note that all customers must be considered and stored in the second output folder (**also the customers with a number of TV series for which they watched all episodes equal to zero**).

Example Part 2

For the sake of simplicity, suppose there are only the following three customers:

- CID1
- CID2
- CID3

For the sake of simplicity, suppose there are only the following two television series:

- SID1
- SID2

Suppose that

- *SID1 is composed of 15 episodes.*
- *SID2 is composed of 34 episodes.*
- *CID1 watched 10 episodes of SID1 and 34 episodes of SID2.*
- *CID2 watched 15 episodes of SID1 and 34 episodes of SID2.*
- *CID3 watched 10 episodes of SID1 and 5 episodes of SID2.*

For this running example, the output stored in the second output folder will be:

```
CID1,1
CID2,2
CID3,0
```

- You do not need to report the imports.
- Suppose both SparkContext sc and SparkSession ss have already been set.