Distributed architectures for big data processing and analytics

July 19, 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 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
# Map input strings to integers
inputADStream = inputDStream\
.map(lambda value: int(value))

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

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

# Define windows, compute the maximum value, and then apply a filter
resADStream = filteredADStream\
.window(30, 10)\
.reduce(lambda v1,v2:max( v1,v2))\
.filter(lambda value: value<10)
```

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

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

#Apply a filter, compute max, define windows
filteredBDStream = inputBDStream\
.filter(lambda value: value>5)\
.reduce(lambda v1,v2:max( v1,v2))\
.window(30, 10)
# Compute the maximum value again and finally apply another filter
resBDStream = filteredBDStream\
.reduce(lambda v1,v2:max( v1,v2))\
.filter(lambda value: value<10)

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

# Part C
# Map input strings to integers and define windows
inputCWindowDStream = ssc.socketTextStream("localhost", 9999)\
.map(lambda value: int(value))\
.window(30, 10)

# Apply a filter and then compute the maximum value
maxCWindowDStream = inputCWindowDStream\
.filter(lambda value: value>5)\
.reduce(lambda v1,v2:max( v1,v2))

# Apply a filter
resCDStream = maxCWindowDStream\
.filter(lambda value: value<10)

# Print the result on the standard output
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** always contain the same integer values.
b) Independently of the content of **inputDStream**, **resADStream** and **resBDStream** always contain 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** always contain 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** always contain 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.

```
HumRDD = sc.textFile("Humidity.txt")

TempRDD = sc.textFile("Temperature.txt")

# Computes the number of lines of Humidity.txt
long numLinesHumidity = HumRDD.count();

# Computes the number of lines of Temperature.txt
long numLinesTemperature = TempRDD.count();

# Print on the standard output the difference between the number of lines of the two files
print("Delta: " + str(numLinesHumidity - numLinesTemperature))

# Create an RDD that contains the intersection of HumRDD and TempRDD
IntersectionRDD = HumRDD.intersection(TempRDD)

# Store the content of IntersectionRDD in the output folder
IntersectionRDD.saveAsTextFile("outputFolder/")

# Print on the standard output the elements in IntersectionRDD
print("Size IntersectionRDD: " + str(IntersectionRDD.count()))
```

Suppose the input files Humidity.txt and Temperature.txt are read from HDFS. Suppose this Spark application is executed only 1 time. Which one of the following statements is true?

a) This application reads the content of Humidity.txt 1 time and the content of Temperature.txt 1 time.
b) This application reads the content of Humidity.txt 3 times and the content of Temperature.txt 3 times.
c) This application reads the content of Humidity.txt 4 times and the content of Temperature.txt 4 times.
d) This application reads the content of Humidity.txt 5 times and the content of Temperature.txt 5 times.

Part II

CoursesOnline is an international company that manages online courses attended by students worldwide. Statistics about the organized courses, lectures, and students are computed based on the following input data files, which have been collected in the company's latest ten years of activity.

- Students.txt
  - Students.txt is a textual file containing information about the students of CoursesOnline. There is one line for each student. The total number of students is greater than 100,000,000. This file is large and you cannot suppose the content of Students.txt can be stored in one in-memory Java/Python variable.
  - Each line of Students.txt has the following format
    - SID,Name,Surname,Country

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

      SID10,Maria,Rossi,Italy

      means that the name and surname of the user with identifier ***SID10*** are ***Maria*** and ***Rossi***, respectively, and the user lives in ***Italy***.

- OnlineCourses.txt
  - OnlineCourses.txt is a textual file containing information about the online courses organized by CoursesOnline. There is one line for each online course. The total number of online courses stored in OnlineCourses.txt is greater than 100,000. This file is large and you cannot suppose the content of OnlineCourses.txt can be stored in one in-memory Java/Python variable.
  - Each line of OnlineCourses.txt has the following format
    - CID,Title,MainTopic

where *CID* is the course's unique identifier, *Title* is the title of the course, and *MainTopic* is the main topic covered by the course.

- For example, the following line

  CID3024,MapReduce and Hadoop,Big data

  means that the course with CID ***CID3024*** is titled ***"MapReduce and Hadoop"*** and covers the ***"Big data"*** topic.

- VideoLectures.txt
  - VideoLectures.txt is a textual file containing information about the video lectures offered by CoursesOnline. There is one line for each video lecture. The total number of video lectures stored in VideoLectures.txt is greater than 2,000,000. This file is large and you cannot suppose the content of VideoLectures.txt can be stored in one in-memory Java/Python variable.
  - Each line of VideoLectures.txt has the following format
    - CODL,Title,Duration,CID

      where *CODL* is the video lecture's unique identifier, *Title* is the video lecture's title, *Duration* is its duration in minutes, and *CID* is the course associated with this video lecture. *Duration* is an integer. Each video lecture is associated with one single course, while each course is associated with/is composed of many video lectures.
    - For example, the following line

      CODL24,Introduction to HDFS,30,CID3024

      means that the video lecture identified by the code ***CODL24*** is titled **"*Introduction to HDFS",*** lasts ***30*** minutes, and is associated with the course with CID ***CID3024***.

- UsersWatchedLectures.txt
  - UsersWatchedLectures.txt is a textual file containing information about who watched which video lectures. A new line is inserted in UsersWatchedLectures.txt every time a student watches a video lecture. UsersWatchedLectures.txt contains the historical data about the latest 10 years. This file is big and you cannot suppose the content of UsersWatchedLectures.txt can be stored in one in-memory Java/Python variable.
  - Each line of UsersWatchedLectures.txt has the following format
    - SID,StartTime,CODL

      where *SID* is the identifier of the student who watched the video lecture identified by *CODL*. The student SID started watching the video lecture *CODL* at *StartTime*. *StartTime* is a timestamp in the format YYYY/MM/DD-HH:MM.
    - For example, the following line

      SID10,CODL24,2023/02/07-20:40

means that the student ***SID10*** watched the video lecture ***CODL24***. He/she started watching the video lecture on ***July 2, 2023,*** at ***20:40***.

Note that each student can watch many video lectures, and each video lecture can be watched by many students. Moreover, **the same student can watch each video lecture several times** (a new line is inserted in UsersWatchedLectures.txt for each visualization). Note that each pair (SID, StartTime) occurs at most one time in UsersWatchedLectures.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of CoursesOnline are interested in performing some statistics.

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

1. *Long courses with many short lectures.* This MapReduce application selects the *long courses* associated with a number of *short video lectures* greater than 10. A course is a *long course* if the sum of the duration of the video lectures of that course is more than 600 minutes. A video lecture is a *short video lecture* if its duration is less than 15 minutes. Store the identifiers (CIDs) of the long courses associated with a number of *short video lectures* greater than 10 in the HDFS output folder (one CID per output line).

Suppose that the input is VideoLectures.txt and has already been set. Suppose that also 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 CoursesOnline asked you to develop one single application to address all the analyses they are interested in. The application has six arguments: the input files Students.txt, OnlineCourses.txt, VideoLectures.txt, and UsersWatchedLectures.txt, and 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. *Main topics with a high percentage of long courses.* The first part of the Spark application selects the main topics characterized by a high percentage of *long courses*. A course is a *long course* if the sum of the duration of the video lectures of that course is more than 600 minutes. This first part of this application selects the main topics with more than 80% of long courses (i.e., a main topic is selected if more than 80% of the courses associated with that topic are long courses). Store the main topics with more than 80% of long courses in the first HDFS output folder (one main topic per output line).

   You can suppose that each course has at least one video lecture.

2. *Distribution of the number of distinct video lectures watched by each student in the years 2021 and 2022 only for the students who are inactive in the year 2023.* The second part of this application considers only the visualizations related to the years 2021, 2022, and 2023 and focuses on the subset of inactive students. A student is considered an *inactive student* if he/she watched no video lectures in the year 2023. For each *inactive student*, computes the number of distinct video lectures he/she watched in the year 2021 and the number of distinct video lectures he/she watched in the year 2022. Store in the second HFDS output folder the identifiers (SIDs) of the inactive users and, for each of them, the number of distinct video lectures he/she watched in the year 2021 and the number of distinct video lectures he/she watched in the year 2022. Specifically, the format of the output lines is

   SID, number of distinct video lectures watched by SID in the year 2021, *number of distinct video lectures watched by SID in the year 2022*

   Note that if an inactive student did not watch any video lectures in 2021 or 2022 (or both), a 0 value must be reported for the year(s) with no watched video lectures. See the example.

   I **remind** you that each user can watch the same video lectures many times.

Example Part 2

For the sake of simplicity, suppose there are only the following four students:

- *SID1*
- *SID2*
- *SID3*
- *SID4*

The following table reports the statistics about the number of distinct video lectures watched by each student in the years 2021 and 2022, and the number of video lectures watched by each student in the year 2023.

SID	Number of distinct video lectures watched in the year 2021	Number of distinct video lectures watched in the year 2022	Number of video lectures watched in the year 2023
SID1	5	1	0
SID2	3	1	5
SID3	0	2	0
SID4	0	0	0

Given this small example, SID1, SID3, and SID4 are inactive students. The second part of this application selects *SID1, SID3, and SID4* and stores in the second output folder the following three lines:

SID1, 5, 1
SID3, 0, 2
SID4, 0, 0

The number of inactive students can be huge (too many to store them in a local variable).

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