Distributed architectures for big data processing and analytics

July 19, 2024

Student ID _____

First Name _____

Last Name _____

The exam is **open book**

Part I

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

1. (2 points) Consider the following Spark application.

```
tempRDD = sc.textFile("Temperature.txt")


# Computes the number of lines of Temperature.txt
numLinesTemp = tempRDD.count()


# Select high temperature values
highTempRDD = tempRDD.map(lambda v: int(v))\
        .filter(lambda v: v>35)\
        .cache()


# Computes the number of high temperatures
numHighTemp = highTempRDD.count()


# Retrieve the maximum high temperature
maxHighTemp = highTempRDD.reduce(lambda v1, v2: max(v1, v2))


# Print on the standard output the three computed values
print("Num lines: " + str(numLinesTemp))
print("Num high temperatures: " + str(numHighTemp))
print("Max high temperature: " + str(maxHighTemp))
```

Suppose the input file Temperature.txt is read from HDFS. Suppose this Spark application is executed only 1 time. Suppose highTempRDD is small enough to be completely cached. Which one of the following statements is **true**?

a) This application reads the content of Temperature.txt 1 time.
b) This application reads the content of Temperature.txt 2 times.
c) This application reads the content of Temperature.txt 3 times.
d) This application reads the content of Temperature.txt 6 times.

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

- ProfilesItaly.txt
  - The text file ProfilesItaly.txt contains the following **4 lines**:
    Luca,Rome
    Luca,Rome
    Carmen,Naples
    Luca,Turin
- ProfilesSpain.txt
  - The text file ProfilesSpain.txt contains the following **2 lines**:
    Carmen,Barcelona
    Laura,Barcelona

Suppose that you are using a Hadoop cluster that can potentially run up to **3 instances of the mapper class** in parallel. Suppose the HDFS block size is 1024MB. Suppose we execute a MapReduce application for Hadoop that analyzes the content of **myFolder**. The content of myFolder is read using the **TextInputFormat** class. Suppose the map phase selects only the lines with a name starting with L and emits, overall, the following key-value pairs (the key part is a selected name while the value part is 1):

(Luca, 1)
(Luca, 1)
(Luca, 1)
(Laura, 1)

Suppose the number of **instances of the reducer class is set to 4** and 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. Specifically, suppose the following pairs are emitted overall by the reduce phase:

(Luca, 3)
(Laura, 1)

Considering all the mapper class instances, how many times is the **map method** invoked?

a) 2
b) 3
c) 4
d) 6

Part II

PoliCourses is an international company that manages online and in-person 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 PoliCourses. 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 student lives in ***Italy***.

- Courses.txt
  - Courses.txt is a textual file containing information about the courses organized by PoliCourses. There is one line for each course. The total number of courses stored in Courses.txt is greater than 200,000. This file is large and you cannot suppose the content of Courses.txt can be stored in one in-memory Java/Python variable.
  - Each line of Courses.txt has the following format
    - CID,Title,Topic

      where *CID* is the course's unique identifier, *Title* is the title of the course, and *Topic* 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.

- Lectures.txt
  - Lectures.txt is a textual file containing information about the lectures offered by PoliCourses. There is one line for each lecture. The total number of lectures stored in Lectures.txt is greater than 3,000,000. This file is large and you cannot suppose the content of Lectures.txt can be stored in one in-memory Java/Python variable.

- Each line of Lectures.txt has the following format
  - NUML,CID,Title,Date,StartingHour,Duration,Recorded
    where the combination *(NUML,CID)* is the lecture's unique identifier, *NUML* is the number of the lecture inside the course, *CID* is the identifier of the course associated with this lecture, *Title* is the lecture's title, *Date* and *StartingHour* are the date and hour at which the lecture is scheduled, *Duration* is its duration in minutes, and *Recorded* specify if the lecture is video recorded (value 'Yes') or not (value 'No'). *Duration* is an integer and represents the lecture's duration in minutes. Each lecture is associated with one single course, while each course is associated with/is composed of many lectures.
  - For example, the following line

    2,CID3024,Introduction to HDFS,2024/01/30,10:00,90,Yes

    means that the lecture identified by the combination ***(2,CID3024)*** is the ***second*** lecture of the course with CID ***CID3024***, is titled **"*Introduction to HDFS",*** was scheduled for ***January, 30, 2024*** at ***10:00***, lasts ***90*** minutes, and is ***video recorded***.

- UsersWatchedRecordedLectures.txt
  - UsersWatchedRecordedLectures.txt is a textual file containing information about who watched which of the recorded lectures. **This file does not contain data about the non-recorded lectures**. A new line is inserted in this file every time a student watches one of the recorded lectures. This file contains historical data about the last 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 UsersWatchedRecordedLectures.txt has the following format
    - SID,StartWatchingTime,NUML,CID
      where *SID* is the identifier of the student who watched the recorded lecture identified by the combination *(NUML,CID)*. The student SID started watching the recorded lecture *(NUML,CID)* at *StartWatchingTime*. *StartWatchingTime* is a timestamp in the format YYYY/MM/DD-HH:MM.
    - For example, the following line

      SID10,2024/02/01-20:40,2,CID3024

      means that the student identified by ***SID10*** watched the recorded lecture identified by the combination ***(2,CID3024)***. He/she started watching the recorded lecture on ***February 1, 2024,*** at ***20:40***.

Note that each student can watch many recorded lectures, and each recorded lecture can be watched by many students. Moreover, **the same student can watch each recorded lecture several times at different starting times** (a new line is inserted in UsersWatchedRecordedLectures.txt for each visualization). The combination of attributes (SID, StartWatchingTime) is the "primary key" of the input

file. Hence, each pair (SID, StartWatchingTime) occurs at most one time in UsersWatchedRecordedLectures.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliCourses are interested in performing some analyses about the managed courses.

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

1. *Courses with all recorded lectures and courses with no recorded lectures.* The application computes the percentage of recorded lectures for each course and selects the courses for which all lectures are recorded (percentage of recorded lectures equal to 100%) and those for which no lectures are recorded (percentage of recorded lectures equal to 0%). Store the result in the output HDFS folder (one select course per output line). For the courses for which all lectures are recorded (100%), store their CIDs and the constant string "All recorded". For the courses without recorded lectures (0%), store their CIDs and the constant string "No recorded lectures".
   Note that there is at least one lecture for each course in Lectures.txt.

   Example
   Suppose, for this toy example, that there are only five courses, identified by the following CIDs: CID1, CID2, CID3, CID4, and CID5.
   Suppose that:
   - CID1 is associated with 10 lectures: 6 recorded lectures and 4 non-recorded lectures.
   - CID2 is associated with 20 lectures: 20 recorded lectures and 0 non-recorded lectures.
   - CID3 is associated with 10 lectures: 9 recorded lectures and 1 non-recorded lecture.
   - CID4 is associated with 5 lectures: 0 recorded lectures and 5 non-recorded lectures.
   - CID5 is associated with 15 lectures: 15 recorded lectures and 0 non-recorded lectures.

   In this case, CID2, CID4, and CID5 are selected.
   CID2 is selected because its percentage of recorded lectures is **100%**.
   CID4 is selected because its percentage of recorded lectures is **0%**.
   CID5 is selected because its percentage of recorded lectures is **100%**.

   The output is as follows:
   CID2,All recorded
   CID4,No recorded lectures
   CID5,All recorded

CID2 is selected because its percentage of recorded lectures is 100%
CID4 is selected because its percentage of recorded lectures is 0%
CID5 is selected because its percentage of recorded lectures is 100%

Suppose that the input is Lectures.txt and it has already been set. Suppose that the name of the output folder has also 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 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 PoliCourses asked you to develop a single Spark-based application based either on RDDs or Spark SQL to address the following tasks. The application takes the paths of the four input files and two output folders (associated with the outputs of the following points 1 and 2, respectively).

1. *Courses with a high percentage of long lectures*. The first part of this application selects the courses with a percentage of long lectures greater than 70%. A lecture is classified as a long lecture if it lasts at least 120 minutes. Store the identifiers (CIDs) of the selected courses in the first output folder (one selected CID per output line).
2. *For each student, the courses for which the student never watched more than one time the course's recorded lectures*. The second part of this application selects, for each student, the courses for which he/she has never watched each of the course's recorded lectures more than one time (i.e., for each student, select the courses for which the student watched from 0 to 1 time each recorded lecture). **For each student, consider only the courses for which the student watched at least one recorded lecture**. Store the result in the second output folder (one of the selected combinations (SID, CID) per output line).

   Example Part 2
   Consider a toy example with a few courses, recorded lectures, and students.
   Suppose there are two courses: CID1 and CID2.
   Suppose that
   - CID1 has 3 recorded lectures identified by (1, CID1), (2, CID1), and (3, CID1)
   - CID2 has 2 recorded lectures identified by (1, CID2) and (2, CID2)

   The following table reports the number of times each student watched each of the recorded lectures.

	Courses Recorded lectures				
	CID1			CID2	
Students	(1, CID1)	(2, CID1)	(3, CID1)	(1, CID2)	(2, CID2)
SID1	1	**2**	0	1	0
SID2	1	1	1	0	1
SID3	**2**	0	**3**	1	**2**
SID4	**2**	**2**	1	0	**3**

   In this case, the output is
   - SID1,CID2
   - SID2,CID1
   - SID2,CID2

The combination (SID1,CID2) **is selected** because student SID1 watched the recorded lectures of CID2 at most one time.
The combination (SID2,CID1) **is selected** because student SID2 watched the recorded lectures of CID1 at most one time.
The combination (SID2,CID2) **is selected** because student SID2 watched the recorded lectures of CID2 at most one time.

- You do not need to write imports. Focus on the content of the main method.
- Suppose both **SparkContext sc** and **SparkSession ss** have already been set.
- **Only if you use Spark SQL**, suppose the first line of all files contains the header information/the name of the attributes. Suppose, instead, there are no header lines if you use RDDs.