Distributed architectures for big data processing and analytics

July 5, 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.

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

(Application A)

```
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)

resADStream = inputDStream\
.map(lambda value: int(value))
.filter(lambda value : value>5)
.window(30, 10)
.reduce(lambda v1,v2: min(v1, v2) )
.filter(lambda value : value<10)


# Print the result
resADStream.pprint()

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

(Application B)

```
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)

resBDStream = inputDStream\
.map(lambda value: int(value))
.reduce(lambda v1,v2: min(v1, v2) )
.filter(lambda value : value>5)
```

```
.window(30, 10)
.reduce(lambda v1,v2: min(v1, v2) )
.filter(lambda value : value<10)


# Print the result
resBDStream.pprint()

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

(Application C)

```
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)

resCDStream = inputDStream\
.map(lambda value: int(value))
.filter(lambda value : value>5)
.reduce(lambda v1,v2: min(v1, v2) )
.filter(lambda value : value<10)
.window(30, 10)
.reduce(lambda v1,v2: min(v1, v2) )
.filter(lambda value : value<10)


# Print the result
resCDStream.pprint()

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

Which one of the following statements is true?

a) Applications A, B, And C are equivalent in terms of returned result, i.e., given the same input they return the same result.

b) Applications A and B are equivalent in terms of returned result, i.e., given the same input they return the same result, while C is not equivalent to the other two applications.

c) Applications A and C are equivalent in terms of returned result, i.e., given the same input they return the same result, while B is not equivalent to the other two applications.

d) Applications B and C are equivalent in terms of returned result, i.e., given the same input they return the same result, while A is not equivalent to the other two applications.

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 four lines:
       Luca,Rome
       Luca,Rome
       Carmen,Naples
       Luca,Turin
   - ProfilesSpain.txt
     - the text file ProfilesSpain.txt contains the following two lines:
       Carmen,Barcelona
       Laura,Barcelona

   Suppose that you are using a Hadoop cluster that can potentially run up to 2 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. Suppose the map phase emits, overall, the following key-value pairs (the key part is a name while the value part is always 1):

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

   Suppose the number of instances of the reducer class is set to 3 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 pair is emitted overall by the reduce phase:

   (Luca, 3)
   (Carmen, 2)
   (Laura, 1)

   Suppose to run the above application once. Which of the following statements is **true**?

   a) Among the 3 instances of the reducer class:
      - The reduce method of the first instance of the reducer class is invoked 2 times.
      - The reduce method of the second instance of the reducer class is invoked 1 time.
      - The reduce method of the third instance of the reducer class is never invoked.

   b) Among the 3 instances of the reducer class:
      - The reduce method of the first instance of the reducer class is invoked 4 times.

- The reduce method of the second instance of the reducer class is invoked 2 times.
- The reduce method of the third instance of the reducer class is never invoked.

c) Among the 3 instances of the reducer class:
   - The reduce method of the first instance of the reducer class is invoked 3 times.
   - The reduce method of the second instance of the reducer class is invoked 2 times.
   - The reduce method of the third instance of the reducer class is invoked 1 time.

d) Among the 3 instances of the reducer class:
   - The reduce method of the first instance of the reducer class is invoked 6 times.
   - The reduce method of the second instance of the reducer class is never invoked.
   - The reduce method of the third instance of the reducer class is never invoked.

Part II

The managers of PoliJob, an international job portal company, asked you to develop some applications to address the analyses they are interested in. The analyses are based on the following input data sets/files.

- JobPostings.txt
  - It is a textual file containing information about the job postings, i.e., positions for jobs that companies are looking for. There is one line for each job posting. The JobID uniquely identifies the job postings. Multiple job postings with the same position (title) can be present. This file is extremely large and you cannot suppose its content can be stored in one in-memory variable.
  - Each line of JobPostings.txt has the following format
    - JobID,Title,Country,Continent,PublicationDate

      where *JobID* is the unique identifier of the job posting, *Title* is the title/type of the open job, *Country* and *Continent* are the country and continent where the job position is open, and PublicationDate is the date of publication of the job posting.

    - For example, the following line

      J1,Software Engineer,IT,Europe,2024/01/24

      means that the job posting with JobID ***J1*** is for a position of ***Software Engineer*** (Title) in ***Italy***, ***Europe***. The job was published on ***January 24, 2024***.

  Note that there can be many job postings for the same title (i.e., many lines of JobPostings.txt can refer to the same title).

- Offers.txt
  - It is a textual file containing information about the job offers, i.e., job offers that were proposed to candidates applying for a Job posting (JobID). There is one line for each offer. OfferID uniquely identifies the offers. The status indicates whether the offer has been accepted or not. If an offer is accepted, then you might have a job contract (i.e., there can be from 0 to 1 line for each offer in Contracts.txt). This file is extremely large and you cannot suppose its content can be stored in one in-memory variable.
  - Each line of Offers.txt has the following format
    - OfferID,JobID,OfferDate,Salary,Status,SSN
      where *OfferID* is the unique identifier of the job offer, *JobID* is the job posting for which the offer is made, *OfferDate* is the date the offer was

made, *Salary* is the offered salary, *Status* indicates whether the offer was accepted or rejected, and SSN is the social security number of the candidate to whom the offer was made.

- For example, the following line

  O101,J1,2024/02/21,97000,Accepted,800-11-2222

  means that the offer ***O101*** was proposed to the candidate with SSN ***800-11-2222*** for job ***J1*** with a salary of ***97000*** euro on ***February 21, 2024***, and the candidate ***accepted*** the offer.

Note that there can be many offers (either accepted or rejected) for the same job posting.

- Contracts.txt
  - It is a textual file containing information about the contracts signed by candidates, i.e., after an offer has been accepted, a contract is typically signed (there are from 0 to 1 contract for each accepted offer). There is one line for each contract, and ContractID uniquely identifies the contracts. This file is extremely large and you cannot suppose its content can be stored in one in-memory variable.
  - Each line of Contracts.txt has the following format
    - ContractID,OfferID,ContractDate,ContractType
      where *ContractID* is the identifier of the contract, *OfferID* is the identifier of the job offer for which the contract is signed, *ContractDate* is the date of the signature of the contract, and *ContractType* the type of contract (Full time, Part time, etc.).
    - For example, the following line

      C201,O101,2024/03/01,Full-time

      means that the contract with id ***C201*** associated with offer ***O101*** is ***full time*** and was signed on ***March 1, 2024***.

Note that there are from 0 to 1 contracts for each accepted offer. No contracts for rejected offers.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliJob are interested in performing some analyses about job postings.

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

1. *Titles associated with many job postings in at least two different European countries*. The application selects the titles associated with at least 30 job postings per country in at least 2 different European countries. Store the selected titles in the output HDFS folder (one title per output line). Note: The number of distinct countries is 100 (i.e., it is small).

Suppose that the input is JobPostings.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 PoliJob asked you to develop a single Spark-based application based on RDDs or Spark SQL to address the following tasks. The application takes the paths of the three input files and two output folders (associated with the outputs of the following points 1 and 2, respectively).

1. *Job postings with many offers in 2024 and a number of rejected offers greater than the number of accepted ones in 2024.* Considering only the offers made from January 1, 2024 (referring to OfferDate), the first part of this application selects the job postings with at least 10 offers (considering all offers in 2024: accepted and rejected) and a number of rejected offers greater than the number of accepted offers (always considering only 2024). Store the JobIDs of the selected job postings and the number of offers for each job posting in 2024 (considering all offers in 2024: accepted+rejected) in the first output folder. Each output line contains one of the selected job postings and its number of offers in 2024.

2. *For each country, select the job titles with a high percentage of accepted offers that are not associated with signed contracts from the year 2000.* The second part of this application selects for each country the job titles with more than 50% of the accepted offers not associated with a contract in at least three different years (not necessarily consecutive), considering the publication date of the job posting (PublicationDate) as the reference year. Consider only the job postings published from 2000 for this second part of the task. Store, in the second HDFS output folder, the selected job titles, the associated countries, and the number of years with more than 50% of the accepted offers not associated with a contract (one job title, country, number of years with more than 50% of the accepted offers not associated with a contract per output line).

   Example Part 2
   Consider a toy example that contains only 12 job postings from the year 2000.
   - *J1,Software Engineer,IT,Europe,2024/01/24*
   - *J2,Software Engineer,IT,Europe,2023/01/24*
   - *J3,Software Engineer,IT,Europe,2023/02/24*
   - *J4,Software Engineer,IT,Europe,2010/01/24*
   - *J5,Software Engineer,IT,Europe,2004/01/24*
   - *J6,Data Engineer,IT,Europe,2020/03/04*
   - *J7,Data Engineer,IT,Europe,2019/03/04*

- *J8,Data Engineer,IT,Europe,2018/03/04*
- *J9,Data Engineer,ES,Europe,2018/05/24*
- *J10,Data Scientist,FR,Europe,2024/01/24*
- *J11,Data Scientist,FR,Europe,2020/02/02*
- *J12,Data Scientist,FR,Europe,2018/03/04*

Suppose these are the statistics, computed from the input files:

Title	Country	Year	Num. of **accepted offers** (considering all offers and job postings related to Title and Country in Year)	Num. of **accepted offers not associated with a contract** (considering all offers related to Title and Country in Year)	**Percentage** of **accepted offers not associated with a contract** (considering all offers related to Title and Country in Year)
Software Engineer	IT	2024	10	6	60%
		2023	2	2	100%
		2010	5	4	80%
		2004	1	0	0%
Data Engineer	IT	2020	1	1	100%
		2019	3	2	66.6%
		2018	2	2	100%
Data Engineer	ES	2018	1	0	0%
Data Scientist	FR	2024	8	6	75%
		2020	2	2	100%
		2018	3	1	33.3%

In this case, the output is

- Software Engineer, IT, 3
- Data Engineer, IT, 3

The combination (Software Engineer, IT) **is selected** because there are at least three years (specifically, 2010, 2023, and 2024) with the percentage of accepted offers not associated with a contract greater than 50% for that combination.

The combination (Data Engineer, IT) **is selected** because there are at least three years (specifically, 2018, 2019, and 2020) with the percentage of accepted offers not associated with a contract greater than 50% for that combination.

The combination (Data Engineer, ES) is **not selected** because there are no years with the percentage of accepted offers not associated with a contract greater than 50% for that combination.

The combination (Data Scientist, FR) is **not selected** because there are only two years (specifically, 2020 and 2024) with the percentage of accepted offers not associated with a contract greater than 50% for that combination.

- You do not need to write Java 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.