Distributed architectures for big data processing and analytics

June 27, 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 application.

```
# Read input file
inputRDD = sc.textFile("TemperatureReadings.txt")

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

# Print on the standard output of the driver the total number of input lines
print("Total number of lines: " + str(inputRDD.count()))

# Print on the standard output of minimum temperature
print("Min. temperature: " + str(tempsRDD.reduce(lambda t1,t2: min(t1,t2))))

# Print on the standard output of maximum temperature
print("Max. temperature: " + str(tempsRDD.reduce(lambda t1,t2: max(t1,t2))))

# Select high temperatures
highTempsRDD = tempsRDD.filter(lambda temp: temp>35)

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

# Select low temperatures
lowTempsRDD = tempsRDD.filter(lambda temp: temp<-15)

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

Suppose the input file TemperatureReadings.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 TemperatureReadings.txt 1 time

b) This application reads the content of TemperatureReadings.txt 3 times

c) This application reads the content of TemperatureReadings.txt 5 times

d) This application reads the content of TemperatureReadings.txt 8 times

2. (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(30, 10)\
.map(lambda value: int(value))

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

# Compute the maximum value and then a filter
resADStream = filteredADStream.reduce(lambda v1,v2:max( v1,v2))\
.filter(lambda value: value<10)

# 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, 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
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
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**.

Part II

PoliDataCenters is a company that manages several data centers located around the world. The staff of PoliDataCenters logs data about the managed servers, the available patches for each operating system, and the applied patches. The analyses of interest are based on the following input data sets/files.

- Servers.txt
  - Servers.txt is a text file containing the list of servers managed by PoliDataCenters. Each line of Servers.txt is associated with one server. PoliDataCenters manages millions of servers.
  - Each line of Servers.txt has the following format
    - SID,OperatingSystem,Model

where *SID* is the unique identifier of the server while *OperatingSystem* is its operating system and *Model* is its model.

- For example, the following line

  S10,Ubuntu6,SunUltraServer1

  means that the server identified by the SID ***S10*** has the operating system ***Ubuntu6*** and is a ***SunUltraServer1*** server.

- Patches.txt
  - Patches.txt is a textual file containing the information about the patches of more than 100 operating systems. There are millions of patches in this file.
  - Each line of Patches.txt has the following format
    - PID,ReleaseDate,OperatingSystem

      where *PID* is the patch identifier, *ReleaseDate* is the date on which the patch was released, and *OperatingSystem* is the operating system for which the patch was released.

    - For example, the following line

      PIDW10_22,2022/01/18,Ubuntu6

      means that the patch with id ***PIDW10_22***, which is a patch for ***Ubuntu6***, was released on ***January 18, 2022***.

- AppliedPatches.txt
  - AppliedPatches.txt is a textual file containing the information about which patches were applied on each server in the last 30 years.
  - Each line of AppliedPatches.txt has the following format
    - PID,SID, Date

      where *PID* is the identifier of the applied patch, *SID* is the identifier of the server on which the patch PID was applied, and Date is the date associated with the application of PID on SID.
      Each line of AppliedPatches.txt is uniquely identified by the primary key (PID,SID). Each patch can be applied to many servers and each server can be associated with many patches but the same combination (PID,SID) occurs at most one time in AppliedPatches.txt.
      Moreover, the data are correct and hence each server was patched only with patches associated with its operating system.

    - For example, the following line

      PIDW10_22,S10,2022/02/21

      means that the patch with id ***PIDW10***_22 was applied on the server with SID ***S10*** on ***February 21, 2022***.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliDataCenters are interested in performing some analyses about the applied patches.

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

1. *Last applied patch in the year 2022 for each server.* The application considers only the servers to which it was already applied at least one patch in the year 2022, and selects for each of those servers the PID of the last applied patch in the year 2022 (the last one in terms of application Date). Store in the output HDFS folder for each server its SID and the PID of the last applied patch (one pair (SID,PID) per output line).

Suppose that the input is AppliedPatches.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

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 PoliDataCenters asked you to develop one single application to address all the analyses they are interested in. The application has five arguments: the input files Servers.txt, Patches.txt, and AppliedPatches.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. *Server(s) with the maximum number of applied patches in the year 2022.* This first part of the application considers only the servers to which it was applied at least one patch in the year 2022, and selects the server(s) with the maximum number of applied patches in the year 2022. For the selected server(s), store in the first HDFS output folder its identifier (SID) and its operating system.
   Pay attention that many servers can be associated with the maximum number of applied patches in the year 2022. Store SID and the operating system of all the selected servers (one combination (SID, operating system) per output line).

2. *Up-to-date servers.* This second part of the application selects up-to-date servers. A server is up-to-date if all the patches available for its operating system were applied to it. The identifiers (SIDs) of the selected servers are stored in the second HDFS output folder (one SID per output line).
   Pay attention. Also the servers with an operating system for which there are no patches are considered up-to-date and must be selected and stored in the second output folder. For instance, suppose that server SID10 has the operating system Ubuntu30 and suppose that there are no patches for Ubuntu30 in Patches.txt. SID10 is stored in the second output folder because it is up-to-date.

   Examples Point 2
   - *First example.* Suppose that there are three servers: SID53, SID4, and SID20. Suppose that Windows10 is installed on SID53 and SID4 and Ubuntu10 is installed on SID20. Suppose that the total number of available patches for Windows10 is 20 and for Ubuntu10 is 34. Suppose the number of patches applied on SID53 is 20, the number of patches applied on SID4 is 15, and the number of patches applied on SID20 is 20. Only server SID53 is selected because it has all the patches of its operating system.
   - *Second example.* Suppose that there are three servers: SID10, SID50, and SID100. Suppose that Windows9 is installed on SID10 and SID50 and Ubuntu12 is installed on SID100. Suppose that the total number of available patches for Windows9 is 20 and for Ubuntu12 is 0 (no patches available). Suppose the number of patches applied on SID10 is 20, the number of patches applied on SID50 is 20, and the number of patches applied on SID100 is 0. All these three servers are up-to-date and their SIDs (SID10, SID50, and SID100) are stored in the second output folder.

- **Write your code on your papers.**
- You do not need to write imports. Focus on the content of the main method.
- Suppose both **SparkContext sc** and **SparkSession ss** have been already set.
- Suppose the following variables have been already set:
  - serversPath='Servers.txt', patchesPath='Patches.txt', appPatches='AppliedPatches.txt', output1='outPart1/', output2='outPart2/'