Distributed architectures for big data processing and analytics

June 27, 2025

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 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>10)\
.window(30, 10)\
.filter(lambda value : value>15)\
.reduce(lambda v1,v2: max(v1, v2) )


# 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))\
.filter(lambda value : value>10)\
.reduce(lambda v1,v2: max(v1, v2) )\
```

```
.window(30, 10)\
.filter(lambda value : value>15)\
.reduce(lambda v1,v2: max(v1, v2) )


# 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>15)\
.window(30, 10)\
.filter(lambda value : value>10)\
.reduce(lambda v1,v2: max(v1, v2) )


# 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 following Spark application.

```
# Read the input file containing student grades
# The format of each line is: StudentID,CourseName,Grade
gradesRDD = sc.textFile("Grades.txt")


# Count the total number of grade records in the file
totalGradesValue = gradesRDD.count()


# Select the grades of the Computer Science course
# and parse the grade as an integer.
CSGradesRDD = gradesRDD \
                        .filter(lambda line: line.split(',')[1] == 'Computer Science') \
                        .map(lambda line: int(line.split(',')[2]))


# Cache CSGradesRDD
CSGradesRDDCached = CSGradesRDD.cache()


# Compute the max grade
maxGradeValue = CSGradesRDDCached.reduce(lambda v1, v2: max(v1, v2))


# Select the grades equal to maxGradeValue and
# count the number of students who passed the exam with maxGradeValue
numMaxGrade = CSGradesRDDCached \
                        .filter(lambda grade: grade==maxGradeValue) \
                        .count()


# Store the content of CSGradesRDDCached
CSGradesRDDCached.saveAsTextFile('output/')
```

Suppose the input file Grades.txt is read from HDFS and this Spark application is executed only **one** time. Suppose also that the content of CSGradesRDD is small enough to be completely stored in memory when cached.

Which one of the following statements is **true**?

a) The file Grades.txt is read from HDFS 1 time.
b) The file Grades.txt is read from HDFS 2 times.
c) The file Grades.txt is read from HDFS 3 times.
d) The file Grades.txt is read from HDFS 4 times.

Part II

The managers of ShopInsight, an international e-commerce platform, have asked you to develop some applications to support the business analyses they are interested in. These analyses are based on the following input datasets/files.

- Users.txt
  - This file contains information about the profiles of users registered on the platform. **Each line represents a user**. This file is extremely large and you cannot suppose its content can be stored in one in-memory variable.
  - Each line in Users.txt has the following format:
    - UserID,Age,Gender,Country,UserType
    - UserID is the unique identifier of the user
    - Age is the age of the user
    - Gender is the gender of the user
    - Country is the country of residence of the user
    - UserType indicates the user category (it assumes four values: Regular, Premium, Business, Guest)
  - Example
    - U45,32,F,IT,Premium
    - It means that user U45 is a 32 years old female from Italy, and is classified as a Premium user (i.e., UserType is Premium).

- Products.txt
  - This file contains information about the products available on the platform. **Each line represents a product**. This file is large and you cannot suppose its content can be stored in one in-memory variable.
  - Each line in Products.txt has the following format:
    - ProductID,Category,Brand,Price
    - ProductID is the unique identifier of the product
    - Category is the product category (e.g., Electronics, Clothing, etc.)
    - Brand is the brand of the product
    - Price is the price of the product in euros
  - Example
    - P789,Electronics,Sony,499.99
    - It means that product P789 is an electronics product from Sony (i.e., its brand is Sony) and its price is €499.99.

- Purchases.txt
  - This is a textual file containing information about purchases made by users. **Each line represents a purchase**. This file is extremely large and you cannot suppose its content can be stored in one in-memory variable.
  - Each line in Purchases.txt has the following format:
    - PurchaseID,UserID,ProductID, PurchaseDate

        - PurchaseID is the unique identifier of the purchase
        - UserID is the identifier of the user who made the purchase
        - ProductID is the identifier of the purchased product
        - PurchaseDate is the date when the purchase occurred (date format: YYYY/MM/DD)
    - Example
        - S123,U45,P789,2024/11/15
        - It means that user U45 purchased item P789 on November 15, 2024. The purchase is identified by the value S123.
    - Note that each user can purchase the same product multiple times and many products on the same date.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of ShopInsight are interested in performing some analyses about the products.

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

1. *Brands that are associated with both the Musical Instruments and the Motorcycles categories.* The application considers only the products of the categories Musical Instruments and Motorcycles and selects the brands with at least one product in both categories. Store the selected brands in the output folder (one selected brand per output line).

Suppose that the input is Products.txt and it has already been set. Suppose that the name of the output folder has also already been set.

Toy example

Consider a toy example that contains only five brands: Brand 1, Brand 2, Brand 3, Brand 4, and Brand 5.

Suppose that

- Brand 1 has 10 products in the Musical Instruments category and 15 products in the Motorcycles category.
- Brand 2 has 1 product in the Musical Instruments category and 1 product in the Motorcycles category.
- Brand 3 has 10 products in the Musical Instruments category and no products in the Motorcycles category.
- Brand 4 has no products in the Musical Instruments category and no products in the Motorcycles category.

- Brand 5 has no products in the Musical Instruments category and 1 product in the Motorcycles category.

The expected output for this toy example is as follows:

- Brand1
- Brand2

- 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 (19 points)

The managers of ShopInsight 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. *For each user type, total number of purchases from 2010 to 2020 made by Italian users*. The first part of this application considers only Italian users and the purchases from 2010 to 2020. It computes, for each user type, the total number of purchases made by the Italian users from 2010 to 2020. Store the result in the first output folder. The output contains one line for each user type. The output format is as follows:
   user type, total number of purchases made by the Italian users of this user type from 2010 to 2020.

   Note. Suppose there is at least one purchase of at least one Italian user for each user type from 2010 to 2020, **i.e., no user type is associated with zero purchases made by Italian users from 2010 to 2020**.

   Output example Part 1
   Suppose that
   - The total number of purchases from 2010 to 2020 made by Italian users associated with the user type Regular is 1501
   - The total number of purchases from 2010 to 2020 made by Italian users associated with the user type Premium is 534
   - The total number of purchases from 2010 to 2020 made by Italian users associated with the user type Business is 10123
   - The total number of purchases from 2010 to 2020 made by Italian users associated with the user type Guest is 467

   The output is as follows
   - Regular,1501
   - Premium,534
   - Business,10123
   - Guest,467

2. *Number of purchases in 2023 for each Italian user who did not make purchases from 2024*. The second part of this application considers only the Italian users without purchases from 2024. Considering only that subset of users, the second part of this application computes the number of purchases in 2023 for each of those users. **The value for each user must be returned even when it is zero**. The result is stored in the second output folder. The output contains one line for each Italian user without purchases from 2024. The output format is as follows:
   UserID, number of purchases made by UserID in 2023

   Example Part 2

Consider a toy example that contains only four Italian users: U10, U12, U34, and U45.

Suppose that

- U10 made **no purchases from 2024** and 5 purchases in 2023.
- U12 made **no purchases from 2024** and 0 purchases in 2023.
- U34 made **no purchases from 2024** and 3 purchases in 2023.
- U45 made **10 purchases from 2024** and 10 purchases in 2023.

The output of the second part is as follows for this toy example:

- U10,5
- U12,0
- U34,3

U45 is no part of the result because that user made some purchases from 2024.

- 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.