Distributed architectures for big data processing and analytics

February 16, 2026

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)


# Process the DStream associated with the TCP socket localhost:9999
resDStreamA = ssc.socketTextStream("localhost", 9999)\
                .map(lambda s: 1)\
                .count()\
                .window(20, 10)\
                .count()\
                .filter(lambda v: v>0)


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

# Start the computation
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)
```

```
# Process the DStream associated with the TCP socket localhost:9999
resDStreamB = ssc.socketTextStream("localhost", 9999)\
                .map(lambda s: 1)\
                .reduce(lambda v1, v2: v1+v2)\
                .window(20, 10)\
                .reduce(lambda v1, v2: v1+v2) \
                .filter(lambda v: v>0)


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

# Start the computation
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)


# Process the DStream associated with the TCP socket localhost:9999
resDStreamC = ssc.socketTextStream("localhost", 9999)\
                .map(lambda s: 1)\
                .window(20, 10)\
                .count()\
                .filter(lambda v: v>0)


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

# Start the computation
ssc.start()
ssc.awaitTerminationOrTimeout(360)
ssc.stop(stopSparkContext=False)
```

Suppose each batch of the input stream contains at least one input string. Which one of the following statements is true?

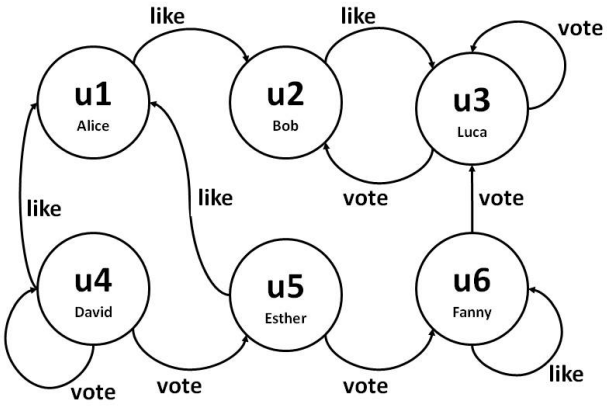
a) Independently of the content of the input stream, **resDStreamA** and **resDStreamB** always contain the same values, while **resDStreamC** may contain different values with respect to resDStreamA and resDStreamB.

b) Independently of the content of the input stream, **resDStreamA** and **resDStreamC** always contain the same values, while **resDStreamB** may contain different values with respect to resDStreamA and resDStreamC.

c) Independently of the content of the input stream, **resDStreamB** and **resDStreamC** always contain the same values, while **resDStreamA** may contain different values with respect to resDStreamB and resDStreamC.

d) None of the other answers is correct.

2. (2 points) Consider the following graph and suppose g is its instantiation in GraphFrame.
   - Schema of the vertexes: ["id", "name"]
   - Schema of the edges: ["src", "dst", "relationship"]



Suppose the following command is executed on g:

motifs = g.find("(v1)-[]->(v2); !(v3)-[]->(v2)")

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

a) One of the rows stored in the Dataframe motifs is

v1	v2	v3
[u4, David]	[u5, Esther]	[u6, Fanny]

b) One of the rows stored in the Dataframe motifs is

v1	v2	v3
[u5, Esther]	[u1, Alice]	[u4, David]

c) One of the rows stored in the Dataframe motifs is

v1	v2	v3
[u6, Fanny]	[u3, Luca]	[u3, Luca]

d) One of the rows stored in the Dataframe motifs is

v1	v2	v3
[u6, Fanny]	[u3, Luca]	[u2, Bob]

Part II

SocialCompany is an international company that manages a location-based social network worldwide. The statistics about users, check-ins, and POIs are computed from the following input data files collected over the last twenty-five years.

- Users.txt
  - Users.txt is a textual file containing information about the users of SocialCompany. There is one line for each user. The total number of users exceeds 200,000,000. Users.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
  - Each line of Users.txt has the following format
    - CodU,Name,Surname,City,Country

      where *CodU* is the unique user identifier, *Name* and *Surname* are the user's name and surname, respectively, while *City* and Country are the user's city and country of residence, respectively.
    - For example, the following line

      U10,Paolo,Garza,Carmagnola,Italy

      means that the name and surname of the user with CodU identifier ***U10*** are ***Paolo*** and ***Garza***, respectively, and the user lives in ***Carmagnola (Italy)***.

- POIs_SocialNetwork.txt
  - POIs_SocialNetwork.txt is a textual file containing information about the points of interest (POIs) worldwide. There is one line for each POI. The total number of POIs stored in POIs_SocialNetwork.txt exceeds 400,000,000. POIs_SocialNetwork.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
  - Each line of POIs_SocialNetwork.txt has the following format
    - POIID,name,latitude,longitude,POICountry

      where *POIID* is the POI's unique identifier, *Name* is its name, *latitude* and *longitude* are its geolocation, and *POICountry* is the country where that POI is located.
    - For example, the following line

POI23,Egyptian Museum,45.0684433,7.6844128,Italy

means that the POI with POIID ***POI23*** is named the ***Egyptian Museum*** and is located in ***Italy*** at the position (latitude = ***45.0684433****,* longitude = ***7.6844128****)*.

- Checkins.txt
  - Checkins.txt is a textual file containing information about check-ins made by users using the Social Network. There is one line for each check-in. The total number of check-ins recorded in Checkins.txt exceeds 500,000,000. Checkins.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
  - Each line of Checkins.txt has the following format
    - CodU,Timestamp,POIID

      where *CodU* is the identifier of the user who checked in to the POI identified by *POIID* at time *Timestamp*. *Timestamp* is a timestamp in the format YYYY/MM/DD-HH:MM.
    - For example, the following line

      U10,2025/11/17-08:04,POI45

      means that the user ***U10*** checked in to the POI ***POI45*** on ***November 17, 2025,*** at ***08:04****.*

Note that each user can check in many POIs at different timestamps, and each POI can be checked in by many users at the same timestamp. Moreover, **the same user may check in to a POI several times** (a new line with a different Timestamp is inserted in Checkins.txt for each check-in to the same POI by the same user). Note that the combination (CodU, Timestamp) is the primary key in Checkins.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of SocialCompany are interested in analyzing check-ins made by users.

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

1. *Users with the highest number of check-ins in 2024.* This application selects the user(s) who made the maximum number of check-ins in 2024. If there are many users associated with the maximum number of check-ins, all these users must be selected and stored in the output folder. Suppose that the set of users associated with the maximum number of check-ins is small and can be stored in a local Java variable.

   Output format (one line for each selected user identifier):
   CodU

Suppose that the input is Checkins.txt and has already been set. Suppose that 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 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 SocialCompany asked you to develop a single Spark-based application, either using RDDs or Spark SQL, to address the following tasks. The application inputs are the paths to the input files. The outputs are stored in two output folders (associated with the outputs of Parts 1 and 2, respectively).

1. *Top-10 users based on the number of distinct checked-in POIs in 2023*. The first part of this application considers only 2023 check-ins and selects the top 10 users based on the number of distinct POIs each user checked in during 2023. Store the result in the first HDFS output folder. Specifically, store one of the top-10 selected users per output line. For each of the selected users, store the CodU.

   Output format of each output line (first part output format):

   CodU

2. *Italian users who are “superstars” of many POIs in 2024 and have a few check-ins in 2023*. The second part of this application selects the Italian users who are “superstars”

of at least 50 POIs in 2024 and who made less than 100 check-ins during 2023 (including those users with 0 check-ins in 2023). A user is a superstar of a POI in 2024 if that user has checked in to that POI at least 1000 times in 2024. Store the result in the second HDFS output folder. The output contains one output line for each of the selected users, and the output format is as follows (second part output format):

CodU, Number of POIs of which CodU is a superstar in 2024

Note. Some users never checked in during 2023. These users satisfy the second part of the condition (i.e., users who have made less than 100 check-ins during 2023).

Example for the second part.

In this small example, suppose there are only four Italian users. The identifiers of these Italian users are U1, U2, U3, and U4.

Suppose that U1 is a superstar of 60 POIs based on the check-ins made in 2024, and has made 5 check-ins in 2023.

Suppose that U2 is a superstar of 70 POIs based on the check-ins made in 2024, and has made 200 check-ins in 2023.

Suppose that U3 is a superstar of 52 POIs based on the check-ins made in 2024, and has made no check-ins in 2023.

Suppose that U4 is a superstar of 10 POIs based on the check-ins made in 2024, and has made 10 check-ins in 2023.

The second output folder must contain the following lines in this case:

U1,60
U3,50

- You do not need to write imports. Focus on the content of the main method.
- **Only if you use Spark SQL**, suppose the first line of each file contains the header information/the name of the attributes. Suppose, instead, there are no header lines if you use RDDs.
- Suppose both Spark Context ***sc*** and SparkSession ***spark*** have already been set.
- Please **comment** your solution by stating the meaning of the fields you intend to process with each instruction, e.g., key=(product id, date), value=(category, year)