Big data processing and analytics

February 10, 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 application.

```
# Define an accumulator. Initialize it to 0
nonItalianCitiesAcc = sc.accumulator(0)

# Create an RDD from a local Python list and cache it.
# The RDD is small and can be entirely stored in the main memory of the servers,
# i.e., it can be completely cached.
citiesRDD= sc.parallelize([(1, "Turin", "IT"),
                           (2, "Rome", "IT"),
                           (3, "Florence", "IT"),
                           (4, "Paris", "FR"),
                           (5, "London", "UK")])\
          .cache()

# Create a second RDD from a local Python list and cache it.
# The RDD is small and can be entirely stored in the main memory of the servers,
# i.e., it can be completely cached.
profilesRDD = sc.parallelize([("U1", "Paolo", 2),
                              ("U2", "Luca", 2),
                              ("U3", "William",1),
                              ("U4", "John", 5)])\
          .cache()

# Filtering function
def italianCities(tupleCity):
    country=tupleCity[2]

    if (country=="IT"):
        return True
    else:
        # Increment the accumulator/the number of non-Italian cities and return False
        nonItalianCitiesAcc.add(1)
        return False
```

```
# Select Italian cities
italianCitiesRDD=citiesRDD.filter(italianCities)

# Print on the standard output the number of non-Italian cities
print("non-Italian cities: ", nonItalianCitiesAcc.value)

# Compute the number of Italian cities and print it on the standard output
numberOfItalianCities=italianCitiesRDD.count()
print("Italian cities: ", numberOfItalianCities)


# Map to (CityID, City name)
cityID_NameRDD = italianCitiesRDD.map(lambda t: (t[0], t[1]))

# Map to (CityID, Username)
cityID_UserNameRDD = profilesRDD.map(lambda t: (t[2], t[1]))

# Join and select username and city name of the users living in Italian cities
userName_CityNameRDD = cityID_UserNameRDD.join(cityID_NameRDD).values()


# Extract the city name part and count the number of occurrences of each Italian city
# where there is at least one user
# The computation is based on the countByValue action
citiesOccsPythonDictionary = userName_CityNameRDD.values().countByValue()


# Print on the standard output the content of the Python dictionary
# citiesOccsPythonDictionary
print("Cities - Occurrences: "+ str(citiesOccsPythonDictionary))


# Print in the standard output a final summary
print("***************\nFinal summary\n***************")

# Print on the standard output the number of non-Italian cities
print("non-Italian cities: ", nonItalianCitiesAcc.value)

# Print on the standard output the number of Italian cities
print("Italian cities: ", numberOfItalianCities)

# Print on the standard output the content of the Python dictionary
# citiesOccsPythonDictionary
print("Cities - Occurrences: "+ str(citiesOccsPythonDictionary))
```

Suppose that the contents of citiesRDD and profilesRDD are small enough to fit in main memory when cached.
Which one of the following statements is true?

a) The execution of this application prints on the standard output of the driver the following result
non-Italian cities: 0
Italian cities: 3
Cities - Occurrences: defaultdict(<class 'int'>, {'Turin': 1, 'Rome': 2})
****
Final summary
****
non-Italian cities: 4
Italian cities: 3
Cities - Occurrences: defaultdict(<class 'int'>, {'Turin': 1, 'Rome': 2})

b) The execution of this application prints on the standard output of the driver the following result
non-Italian cities: 0
Italian cities: 3
Cities - Occurrences: defaultdict(<class 'int'>, {'Turin': 1, 'Rome': 2})
****
Final summary
****
non-Italian cities: 2
Italian cities: 3
Cities - Occurrences: defaultdict(<class 'int'>, {'Turin': 1, 'Rome': 2})

c) The execution of this application prints on the standard output of the driver the following result
non-Italian cities: 2
Italian cities: 3
Cities - Occurrences: defaultdict(<class 'int'>, {'Turin': 1, 'Rome': 2})
****
Final summary
****
non-Italian cities: 2
Italian cities: 3
Cities - Occurrences: defaultdict(<class 'int'>, {'Turin': 1, 'Rome': 2})

d) None of the other answers is correct.

2. (2 points) Consider the following MapReduce application for Hadoop.

DriverBigData.java
/* Driver class */
package it.polito.bigdata.hadoop;

```
import ….;

/* Driver class */
public class DriverBigData extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    int exitCode;
    Configuration conf = this.getConf();

    // Define a new job
    Job job = Job.getInstance(conf);

    // Assign a name to the job
    job.setJobName("MapReduce - Question");

    // Set path of the input file/folder for this job
    FileInputFormat.addInputPath(job, new Path("inputFolder/"));

    // Set path of the output folder for this job
    FileOutputFormat.setOutputPath(job, new Path("outputFolder/"));

    // Specify the class of the Driver for this job
    job.setJarByClass(DriverBigData.class);

    // Set job input format
    job.setInputFormatClass(TextInputFormat.class);

    // Set job output format
    job.setOutputFormatClass(TextOutputFormat.class);

    // Set map class
    job.setMapperClass(MapperBigData.class);

    // Set map output key and value classes
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(NullWritable.class);

    // Set reduce class
    job.setReducerClass(ReducerBigData.class);

    // Set reduce output key and value classes
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(NullWritable.class);

    // Set the number of instances of the reducer class to 3
    job.setNumReduceTasks(3);
```

```
    // Execute the job and wait for completion
    if (job.waitForCompletion(true) == true)
      exitCode = 0;
    else
      exitCode = 1;

    return exitCode;
  }

  /* Main of the driver */
  public static void main(String args[]) throws Exception {
    int res = ToolRunner.run(new Configuration(), new DriverBigData(), args);
    System.exit(res);
  }
}
```

MapperBigData.java

```
/* Mapper class */
package it.polito.bigdata.hadoop;
import …;

class MapperBigData extends
            Mapper<LongWritable, // Input key type
                        Text, // Input value type
                        Text, // Output key type
                        NullWritable> { // Output value type

    protected void map(LongWritable key, // Input key type
                        Text value, // Input value type
                        Context context) throws IOException, InterruptedException {
            // If the value starts with "S", emit (Name, NullWritable)
            if (value.toString().startsWith("S"))
                        context.write(new Text(value), NullWritable.get());
    }
}
```

ReduceBigData.java

```
/* Reducer class */
package it.polito.bigdata.hadoop;
import …;

class ReducerBigData extends Reducer<Text, // Input key type
        NullWritable, // Input value type
        IntWritable, // Output key type
        NullWritable> { // Output value type
```

```
    // Define counter
    int counter;

    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        counter = 0;
    }

    protected void reduce(
            Text key, // Input key type
            Iterable<NullWritable> values, // Input value type
            Context context) throws IOException, InterruptedException {
        // Iterate over the set of values and increment counter
        for (NullWritable value : values) {
            counter++;
        }
    }

    @Override
    protected void cleanup(Context context)
            throws IOException, InterruptedException {
        // Emit (counter, NullWritable)
        context.write(new IntWritable(counter), NullWritable.get());
    }
}
```

Suppose that inputFolder contains the files Names1.txt and Names2.txt. Suppose the HDFS block size is 512 MB.

Content of Names1.txt and Names2.txt:

Filename (size and number of lines)	Content
Names1.txt (33 bytes – 5 lines)	Michael Lucas Sophia Sophia Simon
Names2.txt (24 bytes – 4 lines)	Anna Sophia Simon Samuel

Suppose we run the above MapReduce application (note that the input folder is set to inputFolder/).

What is a ***possible*** output generated by running the above application?

a) The content of the output folder is as follows.

-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00000
-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00001
-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00002
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 _SUCCESS

The content of the part files is as follows.

Filename (number of lines)	Content
part-r-00000 (1 line)	2
part-r-00001 (1 line)	0
part-r-00002 (1 line)	4

b) The content of the output folder is as follows.

-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00000
-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00001
-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00002
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 _SUCCESS

The content of the part files is as follows.

Filename (number of lines)	Content
part-r-00000 (1 line)	1
part-r-00001 (1 line)	0
part-r-00002 (1 line)	2

c) The content of the output folder is as follows.

-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00000
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 part-r-00001
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 part-r-00002
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 _SUCCESS

The content of the part files is as follows.

Filename (number of lines)	Content
part-r-00000 (1 line)	3
part-r-00001 – empty file	
part-r-00002 – empty file	

d) The content of the output folder is as follows.

-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00000
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 _SUCCESS

The content of the part file is as follows.

Filename (number of lines)	Content
part-r-00000 (1 line)	6

Part II

SocialCompany is an international company that manages two location-based social networks worldwide. The social networks are named “Social Network A” and “Social Network B”. Users can have subscribed to one or both social networks. The statistics about users, check-ins, and POIs are computed from the following input data files collected over the last twenty years.

- Users.txt
  - Users.txt is a textual file containing information about the users of SocialCompany. It includes users of both location-based social networks. 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_SNA.txt
  - Checkins_SNA.txt is a textual file containing information about check-ins made by users using the Social Network A app. There is one line for each check-in made using the Social Network A app. The total number of check-ins recorded in Checkins_SNA.txt exceeds 500,000,000. Checkins_SNA.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
  - Each line of Checkins_SNA.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*** using the Social Network A app.

  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_SNA.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_SNA.txt.

- Checkins_SNB.txt
  - Checkins_SNB.txt is similar to Checkins_SNA.txt, but it stores the check-ins related to Social Network B. Social Network B.txt is a textual file containing information about check-ins made by users using the Social Network B app. There is one line for each check-in made using the Social Network B app. The total number of check-ins recorded in Checkins_SNB.txt exceeds 500,000,000. Checkins_SNB.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
  - Each line of Checkins_SNB.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

      U10,2024/02/16-18:00,POI56

      means that the user ***U10*** checked in to the POI ***POI56*** on ***February 16, 2024,*** at ***18:00*** using the Social Network B app.

  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_SNB.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_SNB.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of SocialCompany are interested in analyzing check-ins made by users using the Social Network A app.

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

1. *Users for whom there are at least 10 distinct POIs where the user made many check-ins in 2024 using the Social Network A app.* This application selects a user if there are at least 10 distinct POIs where the user has made many check-ins in 2024 using the Social Network A app. A user has made many check-ins in a POI in 2024 using the Social Network A app if that user has made at least 500 check-ins in that POI in 2024 using the Social Network A app. The identifiers of the selected users are stored in the output HDFS folder.

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

Suppose that the input is Checkins_SNA.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 Italian users based on the number of check-ins in 2024 using the Social Network A app*. The first part of this application considers only Italian users and selects the top-10 Italian users based on the number of check-ins made by the Italian users in the year 2024 using the Social Network A app. 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 and city.

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

   CodU,City

2. *Italian users who are "superstars" of many POIs in Social Network A and have a few check-ins in Social Network B*. The second part of this application selects the Italian users who are "superstars" of at least 20 POIs, considering the check-ins using Social Network A app, and who have made less than 10 check-ins using Social Network B app (remember to account those users who never checked-in using Social Network B app, i.e., those with 0 check-ins with Social Network B app). A user is a superstar of a POI if that user has checked in to that POI at least 1000 times using the Social Network A app. 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

   Note. Some users never check in using Social Network B app. These users satisfy the second part of the condition (i.e., users who have made less than 10 check-ins using Social Network B app).

   Example for the second part.

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

   Suppose that U1 is a superstar of 30 POIs based on the check-ins made using the Social Network A app, and has made 5 check-ins using the Social Network B app.

Suppose that U2 is a superstar of 25 POIs based on the check-ins made using the Social Network A app, and has made 15 check-ins using the Social Network B app.

Suppose that U3 is a superstar of 50 POIs based on the check-ins made using the Social Network A app, and has made no check-ins using the Social Network B app.

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

U1,30
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)