Big data processing and analytics

January 27, 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)\
                .reduce(lambda v1, v2: v1+v2)\
                .window(20, 10)\
                .count()


# 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)


# 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()


# 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) Independently of the content of the input stream, **resDStreamA**, **resDStreamB,** and **resDStreamC** always contain the same values.

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

   // Set the 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(IntWritable.class);
    job.setMapOutputValueClass(IntWritable.class);



    // Set the number of reducers to 0 - Map only job
    job.setNumReduceTasks(0);

    // 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
                        IntWritable, // Output key type
                        IntWritable> { // Output value type

    int c1, c2;

    protected void setup(Context context) {
                    c1 = 0;
                    c2 = 0;
    }

    protected void map(LongWritable key, // Input key type
                    Text value, // Input value type
                    Context context) throws IOException, InterruptedException {

            // If the value starts with "S", increment c1. Otherwise, c2.
            if (value.toString().startsWith("S"))
```

```
                c1++;
        else
                c2++;
    }

    protected void cleanup(Context context) throws IOException, InterruptedException {
        // Emit the pair (c1, c2)
        context.write(new IntWritable(c1), new IntWritable(c2));
    }

}
```

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 (34 bytes – 5 lines)	Michael Lucas Sophia Sophia Simons
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 4 set 3 14:00 part-m-00000
-rw-r--r-- 1 paolo paolo 4 set 3 14:00 part-m-00001
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 _SUCCESS
```

The content of the two part files is as follows.

Filename (number of lines)	Content
part-m-00000 (1 line)	3 2
part-m-00001 (1 line)	3 1

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

-rw-r--r-- 1 paolo paolo 4 set 3 14:00 part-m-00000
-rw-r--r-- 1 paolo paolo 4 set 3 14:00 part-m-00001
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 _SUCCESS

The content of the two part files is as follows.

Filename (number of lines)	Content
part-m-00000 (1 line)	2 2
part-m-00001 (1 line)	3 1

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

-rw-r--r-- 1 paolo paolo 4 set 3 14:00 part-m-00000
-rw-r--r-- 1 paolo paolo 4 set 3 14:00 part-m-00001
-rw-r--r-- 1 paolo paolo 0 set 3 14:00 _SUCCESS

The content of the two part files is as follows.

Filename (number of lines)	Content
part-m-00000 (1 line)	6 3
part-m-00001 (1 line)	0 0

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

-rw-r--r-- 1 paolo paolo 4 set 3 14:00 part-m-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-m-00000 (1 line)	6 3

Part II

TourismPolito is an international company that tracks tourists as they visit points of interest (POIs) worldwide. Statistics about tourists and POIs are computed from the following input data files collected over the last thirty years.

- Tourists.txt
  - Tourists.txt is a textual file containing information about the tourists managed by TourismPolito. There is one line for each tourist. The total number of tourists exceeds 300,000,000. Tourists.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
  - Each line of Tourists.txt has the following format
    - CodT,Name,Surname,City,Country

      where *CodT* is the unique tourist identifier, *Name* and *Surname* are the tourist's name and surname, respectively, while *City* and *Country* are the city and country where the tourist resides, respectively.
    - For example, the following line

      T10,Paolo,Garza,Carmagnola,Italy

      means that the name and surname of the tourist with CodT identifier ***T10*** are ***Paolo*** and ***Garza***, respectively, and the tourist lives in ***Carmagnola*** (***Italy***).

- POIs.txt
  - POIs.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.txt exceeds 400,000,000. POIs.txt is large. Its content cannot be stored in one in-memory Java/Python variable.
  - Each line of POIs.txt has the following format
    - POIID,name,category,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,Museum,45.0684433,7.6844128,Italy

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

- Visits.txt
  - Visits.txt is a textual file containing information about who visited each POI and when. There is one line for each visit. The total number of visits recorded in

Visits.txt exceeds 1,000,000,000. Visits.txt is large. Its content cannot be stored in one in-memory Java/Python variable.

- Each line of Visits.txt has the following format
  - CodT,StartTimestamp,POIID,Duration

    where *CodT* is the identifier of the tourist who started visiting the POI identified by *POIID* at the time *StartTimestamp*. *StartTimestamp* is a timestamp in the format YYYY/MM/DD-HH:MM. The visit lasted *Duration* minutes.
  - For example, the following line

    U10,2022/11/07-21:40,POI23,48.5

    means that the tourist ***U10*** started visiting the POI ***POI23*** on ***November 7, 2022,*** at ***21:40***. The visit lasted ***48.5*** minutes.

Note that each tourist can visit many POIs but at different timestamps, and each POI can be visited by many users at the same timestamp. Moreover, **the same tourist may visit each POI several times** (a new line associated with a different StartTimestamp is inserted in Visits.txt for each visit to the same POI by the same tourist). Note that **the combination (CodT, StartTimestamp) is the primary key** in Visits.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of TourismPolito are interested in performing some analyses about POIs.

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

1. *Countries with many POIs but a few POI categories.* The application selects the countries where the number of POIs is greater than 10,000, and their POIs are overall associated with at most 2 categories. Suppose the total number of distinct categories is 1000 (that value can be considered small). The selected countries are stored in the output HDFS folder.

   Output format (one line for each selected country):

   country

Suppose that the input is POIs.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 Driver class must not be reported/implemented.
- 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 TourismPolito 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 three input files. The outputs are stored in two output folders (associated with the outputs of Parts 1 and 2, respectively).

1. *Tourist(s) with the most visits to Italian POIs*. The first part of this application selects the tourist(s) associated with the maximum number of visits to POIs located in Italy. Each line in Visits.txt is a visit. Select all the tourists associated with the maximum number of visits to POIs located in Italy. Store the result in the first HDFS output folder. Specifically, store one of the selected tourists per output line. For each of the selected tourists, store the CodT. Suppose the maximum number of visits to POIs located in Italy is at least one.

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

   CodT

2. *Number of distinct visited categories for each tourist in 2024 by considering Italian POIs.* For each tourist, the second part of this application counts the number of distinct POI categories the tourist visited in 2024, considering only POIs located in Italy. The application returns 0 for the tourists who never visited POIs located in Italy during the year 2024. Store the result in the second HDFS output folder. The number of output lines is equal to the total number of tourists. Specifically, there is one output line for each tourist, and the output format is as follows (second part output format):

CodT, Number of distinct POI categories the tourist visited in 2024 by considering only POIs located in Italy

Note. Some tourists did not visit POIs that are located in Italy during the year 2024. The output is CodT,0 for those tourists.

Example for the second part.

In this small example, suppose there are only three tourists. The identifiers of these tourists are T1, T2, and T3.

In this small example, suppose there are only the following five POIs in Italy:

- POI1. Its category is 'Museum'.
- POI2. Its category is 'Museum'.
- POI3. Its category is 'Restaurant'.
- POI4. Its category is 'Railway station'.
- POI5. Its category is 'Hotel'.

Suppose that T1 visited the following Italian POIs during the year 2024:

- POI1
- POI2
- POI3

Suppose that T2 visited the following Italian POIs during the year 2024:

- POI1
- POI4

Suppose that T3 visited no Italian POIs during the year 2024.

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

- T1,2
- T2,2
- T3,0

- 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)