Distributed architectures for big data processing and analytics

July 11, 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))\
.reduce(lambda v1,v2: min(v1, v2) )\
.filter(lambda value : value>5)\
.window(40, 10)\
.reduce(lambda v1,v2: min(v1, v2) )\
.filter(lambda value : value<10)


# 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>5)\
.window(40, 10)\
.reduce(lambda v1,v2: min(v1, v2) )\
.filter(lambda value : value<10)


# 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>5)\
.reduce(lambda v1,v2: min(v1, v2) )\
.window(40, 10)\
.reduce(lambda v1,v2: min(v1, v2) )\
.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** and **resBDStream** always contain the same integer values, while **resCDStream** may contain different integer values with respect to resADStream and resBDStream.

b) Independently of the content of **inputDStream**, **resADStream** and **resCDStream** always contain the same integer values, while **resBDStream** may contain different integer values with respect to **resADStream** and **resCDStream**.
c) Independently of the content of **inputDStream**, **resBDStream** and **resCDStream** always contain the same integer values, while **resADStream** may contain different integer values with respect to **resBDStream** and **resCDStream**.
d) **resADStream**, **resBDStream**, and **resCDStream** may contain different integer values, i.e., Application A is not equivalent to Application B, and Application A is not equivalent to Application C, and Application B is not equivalent to Application C.

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("Exercise 11/07/2025");

    // 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 reducers to 2
   job.setNumReduceTasks(2);



   // 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 {

                       // Emit the pair (value, NullWritable)
                       context.write(new Text(value), NullWritable.get());
    }
}
```

ReducerBigData.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 numD
    int numD;

    protected void setup(Context context) {
            // Initialize numD
            numD = 0;
    }

    protected void reduce(Text key, // Input key type
                        Iterable<NullWritable> values, // Input value type
                        Context context) throws IOException, InterruptedException {
            // Increment numD if key starts with "D"
            if (key.toString().startsWith("D") == true) {
                        numD++;
            }

    }

    protected void cleanup(Context context) throws IOException, InterruptedException {
            // Emit the pair (numD, NullWritable))
            context.write(new IntWritable(numD), NullWritable.get());
    }
}
```

Suppose that inputFolder contains the files Cities1.txt and Cities2.txt. Suppose the HDFS block size is 1024 MB.

Content of Cities1.txt and Cities2.txt:

Filename	Content
Cities1.txt	Beijing Cairo **Delhi** **Dhaka** **Dortmund** Mexico City Mumbai São Paulo
Cities2.txt	Cairo Chongqing **Delhi** Istanbul Kolkata

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-r-00000
-rw-r--r-- 1 paolo paolo 2 set 3 14:00 part-r-00001
-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 (2 lines)	2 1
part-r-00001 (1 line)	0

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 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 (1 line)	1

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 2 set 3 14:00 part-r-00001
-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)	1

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 2 set 3 14:00 part-r-00001
-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)	4
part-r-00001 (1 line)	0

Part II

PoliCourses is an international company that manages online courses attended by students worldwide. Statistics about the organized courses, lectures, and students are computed based on the following input data files, which have been collected in the company's latest twenty years of activities.

- Students.txt
  - Students.txt is a textual file containing information about the students of PoliCourses. There is one line for each student. The total number of students is greater than 100,000,000. This file is large and you cannot suppose the content of Students.txt can be stored in one in-memory Java/Python variable.
  - Each line of Students.txt has the following format
    - SID,Name,Surname,Country,Continent

      where *SID* is the user's unique identifier, *Name* and *Surname* are his/her name and surname, respectively, *Country* is the country where he/she lives, and *Continent* is the continent where he/she lives.
    - For example, the following line

      SID10,Maria,Rossi,Italy,Europe

      means that the name and surname of the user with identifier ***SID10*** are ***Maria*** and ***Rossi***, respectively, and the student lives in ***Italy***, which is in ***Europe***.

- Courses.txt
  - Courses.txt is a textual file containing information about the courses organized by PoliCourses. There is one line for each course. The total number of courses stored in Courses.txt is greater than 200,000. This file is large and you cannot suppose the content of Courses.txt can be stored in one in-memory Java/Python variable.
  - Each line of Courses.txt has the following format
    - CID,Title,ProductionYear

      where *CID* is the course's unique identifier, *Title* is the title of the course, and *ProductionYear* is the year when the course was produced/recorded.
    - For example, the following line

      CID3024,MapReduce and Hadoop,2004

      means that the course with CID ***CID3024*** is titled ***"MapReduce and Hadoop"*** and was produced/recorded in 2004.

- RecordedLectures.txt
  - RecordedLectures.txt is a textual file containing information about the lectures offered by PoliCourses. There is one line for each lecture. The total number of lectures stored in RecordedLectures.txt is greater than 4,000,000. This file is large and you cannot suppose the content of RecordedLectures.txt can be stored in one in-memory Java/Python variable.
  - Each line of RecordedLectures.txt has the following format

        - LID,CID,Title,Duration
          where *LID* is the lecture's unique identifier, *CID* is the identifier of the course this recorded lecture is part of, *Title* is the lecture's title, and *Duration* is its duration in minutes. *Duration* is an integer and represents the lecture's duration in minutes. Each lecture is associated with one single course, while each course is associated with/is composed of many lectures.
        - For example, the following line
          LID10,CID3024,Introduction to HDFS, 90

          means that the lecture identified by ***LID10*** is part of the course with CID ***CID3024***, is titled **"*Introduction to HDFS",*** and lasts ***90*** minutes.

- StudentsWatchedRecordedLectures.txt
    - StudentsWatchedRecordedLectures.txt is a textual file containing information about who watched the recorded lectures and when. A new line is inserted in this file every time a student watches one of the recorded lectures. This file contains historical data about the last 20 years. This file is big and you cannot suppose the content of StudentsWatchedRecordedLectures.txt can be stored in one in-memory Java/Python variable.
    - Each line of StudentsWatchedRecordedLectures.txt has the following format
        - SID,StartWatchingTime,LID
          where *SID* is the identifier of the student who watched the recorded lecture identified by *LID*. The student SID started watching the recorded lecture *LID* at *StartWatchingTime*. *StartWatchingTime* is a timestamp in the format YYYY/MM/DD-HH:MM.
        - For example, the following line
          SID10,2024/02/01-20:40,LID10

          means that the student identified by ***SID10*** watched the recorded lecture identified by ***LID10****.* He/she started watching the recorded lecture on ***February 1, 2024,*** at ***20:40***.

Note that each student can watch many lectures, and each lecture can be watched by many students. Moreover, **the same student can watch each lecture several times at different starting times** (a new line is inserted in StudentsWatchedRecordedLectures.txt for each visualization). The combination of attributes (SID, StartWatchingTime) is the "primary key" of the input file. Hence, each pair (SID, StartWatchingTime) occurs at most one time in StudentsWatchedRecordedLectures.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

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

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

1. *Number of countries with many students for each continent.* For each continent, the application computes how many countries have at least 1000 students.

   Each output line has the following format:
   Continent,Number of countries with at least 1000 students for this continent

   Suppose that in each continent there is at least one country with at least 1000 students, i.e., do not consider the case (Continent, 0 countries with at least 1000 students).

   Suppose the number of distinct countries is big and the set of distinct countries cannot be stored in one single Java variable.

Suppose that the input is Students.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 two continents (Europe and Asia) and five countries (Italy, France, Spain, Uzbekistan, and Vietnam).

Suppose that

- Suppose that 1200 students live in Italy (Europe).
- Suppose that 2101 students live in France (Europe).
- Suppose that 400 students live in Spain (Europe).
- Suppose that 1100 students live in Uzbekistan (Asia).
- Suppose that 900 students live in Vietnam (Asia).

The expected output for this toy example is as follows:

- Europe,2
- Asia,1

- 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 and RDDs (19 points)

The managers of PoliCourses 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 input files and two output folders (associated with the outputs of the following points 1 and 2, respectively).

1. *Number of distinct lectures of the course CID10 watched by each student in each year from 2015 to 2020*. The first part of this application considers only the visualizations from 2015 to 2020. The first part of this application computes the number of distinct lectures of the course with CID “CID10” watched by each student in each of the years from 2015 to 2020. Store the result in the first output folder. The output contains one line for each combination (student, year), where year ranges from 2015 to 2020. The output format is as follows:

SID,year,number of distinct lectures of the course identified by CID10 watched by this student (SID) in this specific year

Note. Do not consider the combinations (student, year) for which the student watched no lectures of the course CID10 in year, i.e., the combinations (student, year) with no visualizations must not be considered.

Part 1 - Example

For this toy example, suppose there are only two students (SID1 and SID2).

Suppose that

- SID1 watched
  - 3 distinct lectures of the course CID10 in 2015
  - 4 distinct lectures of the course CID10 in 2016
  - 4 distinct lectures of the course CID10 in 2017
  - 5 distinct lectures of the course CID10 in 2018
  - 3 distinct lectures of the course CID10 in 2019
  - 2 distinct lectures of the course CID10 in 2020
- SID2 watched
  - 1 distinct lectures of the course CID10 in 2015
  - 3 distinct lectures of the course CID10 in 2016
  - 2 distinct lectures of the course CID10 in 2017
  - No lectures of the course CID10 in 2018
  - 4 distinct lectures of the course CID10 in 2019
  - No lectures of the course CID10 in 2020

The output stored in the first output folder is as follows.
SID1,2015,3
SID1,2016,4
SID1,2017,4
SID1,2018,5
SID1,2019,3
SID1,2020,2
SID2,2015,1
SID2,2016,3
SID2,2017,2
SID2,2019,4

Note that the combinations (SID2, 2018) and (SID2, 2020) are not considered and are not stored in the first output folder because SID2 watched no lectures of CID10 in those two years.

2. *Course(s) with the maximum number of visualizations in 2024 and the minimum number of visualizations in 2023*. The second part of this application considers only the visualizations related to the years 2023 and 2024. The second part of this application calculates for each course the number of visualizations (each line of

StudentsWatchedRecordedLectures.txt is a visualization) in 2023 and the number of visualizations in 2024. Then, it selects the course(s) associated with the maximum number of visualizations in 2024 and the minimum number of visualizations in 2023. In case of a tie, all courses associated with the maximum value in 2024 and the minimum value in 2023 must be selected and stored in the output folder. If there are no courses that satisfy both constraints, the output folder is empty.
The result is stored in the second output folder (one selected course per output line). The output format is as follows:
CID,Number of visualizations associated with this course in 2024,Number of visualizations associated with this course in 2023

Note that **the case with both values** (the maximum number of visualizations in 2024 and the minimum number of visualizations in 2023) **equal to zero must also be considered.**

Note that the **output** is **empty if there are no courses that satisfy both constraints**.

Part 2 - First example
Consider a toy example with three courses identified by CID1, CID2, and CID3.

Suppose that

- CID1
  - The number of visualizations associated with the course CID1 in 2024 is 3.
  - The number of visualizations associated with the course CID1 in 2023 is 2.
- CID2
  - The number of visualizations associated with the course CID2 in 2024 is 2.
  - The number of visualizations associated with the course CID2 in 2023 is 2.
- CID3
  - The number of visualizations associated with the course CID3 in 2024 is 3.
  - The number of visualizations associated with the course CID3 in 2023 is 0.

The maximum value in 2024 is 3 and the minimum value in 2023 is 0. The output of the second part is as follows for this first toy example:

- CID3,3,0

Part 2 - Second example
Consider a second toy example with three courses identified by CID1, CID2, and CID3.

Suppose that

- CID1
  - The number of visualizations associated with the course CID1 in 2024 is 4.
  - The number of visualizations associated with the course CID1 in 2023 is 2.

- CID2
  - The number of visualizations associated with the course CID2 in 2024 is 4.
  - The number of visualizations associated with the course CID2 in 2023 is 2.
- CID3
  - The number of visualizations associated with the course CID3 in 2024 is 3.
  - The number of visualizations associated with the course CID3 in 2023 is 3.

The maximum value in 2024 is 4 and the minimum value in 2023 is 2. The output of the second part is as follows for this second toy example:

- CID1,4,2
- CID2,4,2

Part 2 - Third example
Consider a third toy example with three courses identified by CID1, CID2, and CID3.

Suppose that
- CID1
  - The number of visualizations associated with the course CID1 in 2024 is 5.
  - The number of visualizations associated with the course CID1 in 2023 is 2.
- CID2
  - The number of visualizations associated with the course CID2 in 2024 is 4.
  - The number of visualizations associated with the course CID2 in 2023 is 2.
- CID3
  - The number of visualizations associated with the course CID3 in 2024 is 3.
  - The number of visualizations associated with the course CID3 in 2023 is 0.

The maximum value in 2024 is 5 and the minimum value in 2023 is 0. The output of the second part is empty for this third example. CID1 is associated with the maximum value in 2024 but is not associated with the minimum value in 2023. CID3 is associated with the minimum value in 2023 but is not associated with the maximum value in 2024.

- You do not need to write 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.