Distributed architectures for big data processing and analytics

July 13, 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 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("Exam 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 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 count
    int count;

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


    protected void reduce(Text key, // Input key type
                Iterable<NullWritable> values, // Input value type
                Context context) throws IOException, InterruptedException {
        // Define and initialize occurrences
        int occurrences = 0;

        // Iterate over the set of values and increment occurrences
        for (NullWritable value: values) {
                occurrences++;
        }

        // Increment count if occurrences is greater than 2
        if (occurrences>2)
```

```
                count++;
    }

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

Suppose that inputFolder contains the files Cities1.txt, Cities2.txt, and Cities3.txt. Suppose the HDFS block size is 1024 MB.

Content of Cities1.txt, Cities2.txt, and Cities3.txt:

Filename	Content
Cities1.txt	Rome Turin
Cities2.txt	Rome Turin
Cities3.txt	Milan Rome Turin

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

What is a ***potential*** output that could be generated by running the MapReduce application reported above?

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 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)	1

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)	2
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)	3
part-r-00001 (1 line)	0

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 files is as follows.

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

2. (2 points) Consider the following Spark application.

```
RDD_A = sc.textFile("A.txt")
```

```
# Select a subset of lines
RDD_B = RDD_A.filter(lambda line: line.split(',')[1] == 'Polito')


# Count the number of elements and print it
v2 = RDD_B.count()

print(v2)


 # Create an RDD of integers
RDD_C = RDD_B.map(lambda line: int(line.split(',')[0]))


# Compute max and min, and their difference
min_Value = RDD_C.reduce(lambda v1, v2: min(v1, v2))

max_Value = RDD_C.reduce(lambda v1, v2: max(v1, v2))


diff_value = max_Value - min_Value


# Print diff_value

print(diff_value)
```

Suppose the input file A.txt is read from HDFS, and this Spark application is executed only **once**.

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

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

Part II

PoliSecurity is a company that monitors the security of servers in big data centers located worldwide. These analyses are based on the following input datasets/files.

- Servers.txt
  - Servers.txt is a text file listing the servers managed by PoliSecurity. Servers.txt contains one line for each server. This file is large. You cannot assume its content can be stored in a single in-memory variable.
  - Each line has the following format:
    - SID,OperatingSystem,IP,Country
    - where
      - SID is the unique server identifier
      - OperatingSystem is the installed operating system (suppose each server is equipped with a single operating system)
      - IP is its IP address
      - Country is the country where the server is located
    - For example, the following line

      S1,Ubuntu10.0,130.192.12.21,Italy

      means that the operating system installed on the server is Ubuntu10.0, the server is identified by the SID S1, its IP address is 130.192.12.21, and the server is in Italy.

- Patches.txt
  - This file contains the information for the released patches. Each line represents a patch. This file is large. You cannot assume its content can be stored in a single in-memory variable.
  - Each line has the following format:
    - PID,ReleaseDate,OperatingSystem,CriticalityLevel
    - where
      - PID is the unique patch identifier
      - ReleaseDate is the date on which the patch was released (YYYY/MM/DD)
      - OperatingSystem is the identifier of the operating system for which the patch was released
      - CriticalityLevel is an integer between 0 and 10 that represents the criticality of the vulnerability associated with the patch (the higher the value, the more critical the vulnerability).
    - For example, the following line

      PID7000,2017/10/01,Ubuntu20.6,9

      means that the patch with PID PID7000, which is a patch for the operating system Ubuntu20.6, was released on October 1, 2017, and its criticality level is 9.

- AppliedPatches.txt
  - This file contains the information about the patches applied to the servers. Each line represents the application of a specific patch to a specific server. This file is large. You cannot assume its content can be stored in a single in-memory variable.
  - Each line has the following format:
    - PID, SID, Timestamp
    - where
      - PID is the patch identifier
      - SID is the server identifier
      - Timestamp is the timestamp at which the patch PID was applied to the server SID (YYYY/MM/DD:HH:MM:SS)
    - For example, the following line

      PID120,SID20,2025/06/01:10:05:00

      means that the patch with PID PID120 has been applied to the server with SID SID20 on June 1, 2025, at 10:05:00.
  - The primary key is the triplet PID, SID, Timestamp. Note that the same patch can be applied more than once on the same server at different timestamps.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliSecurity are interested in performing some analyses about the servers and patches.

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

1. *Select operating systems with an increased number of patches from 2024 to 2025 and the maximum difference between the two years.* This application first identifies the operating systems with a number of patches released in 2025 greater than the number of patches released in 2024 and then, considering only this subset of operating systems, selects the one with the maximum difference between the number of patches released in 2025 and the number of patches released in 2024. In case of a tie, select the first operating system in alphabetical order. Store the result in the output folder.

   The single output line has the following format:
   Operating system,difference between the number of patches released in 2025 and the number of patches released in 2024

Suppose that the input is Patches.txt and it has already been set. Suppose that the name of the output folder has also 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 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 PoliSecurity 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 one output folder (associated with the output of point 2).

1. *Operating systems with an increasing number of released patches.* The first part of this application counts how many operating systems are characterized by a number of patches released in 2025 greater than the number of patches released in 2024. Print the result (i.e., the number of operating systems satisfying the constraint) on the standard output.

2. *Not fully updated servers.* The second part of this application focuses on “not fully updated servers”. A server is considered a “not fully updated server” if there is at least one patch for its operating system that has not been applied to that server. The second part of this application stores the identifiers of the “not fully updated servers” in the output folder (one SID per output line). The output format is as follows:
   SID

   Remind that the same patch can be applied more than once on the same server at different timestamps.

   Suppose that **there is at least one released patch for each operating system**.

   Example Part 2
   Consider a toy example that contains
   - only two operating systems: Ubuntu10 and Ubuntu11
   - only four servers: the server identified by SID1 (with Ubuntu10), the server identified by SID2 (with Ubuntu10), the server identified by SID3 (with Ubuntu10), and the server identified by SID4 (with Ubuntu11)

   Suppose that:
   - There are 100 patches for Ubuntu10 in Patches.txt.
   - There are 50 patches for Ubuntu11 in Patches.txt.

   Suppose that:
   - 100 patches have been applied to the server identified by SID1
   - 50 patches have been applied to the server identified by SID2
   - 0 patches have been applied to the server identified by SID3
   - 50 patches have been applied to the server identified by SID4

   For this toy running example, the servers identified by SID2 and SID3 are “not fully updated servers” because
   - SID2 has an operating system (Ubuntu10) with 100 released patches, but only 50 of them have been applied to SID2
   - SID3 has an operating system (Ubuntu10) with 100 released patches, but none of them have been applied to SID3

Therefore, for this running example, the content of the output folder associated with the second part of this application is:
SID2
SID3

- 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.