Distributed architectures for big data processing and analytics

September 5, 2025

Student ID _____

First Name _____

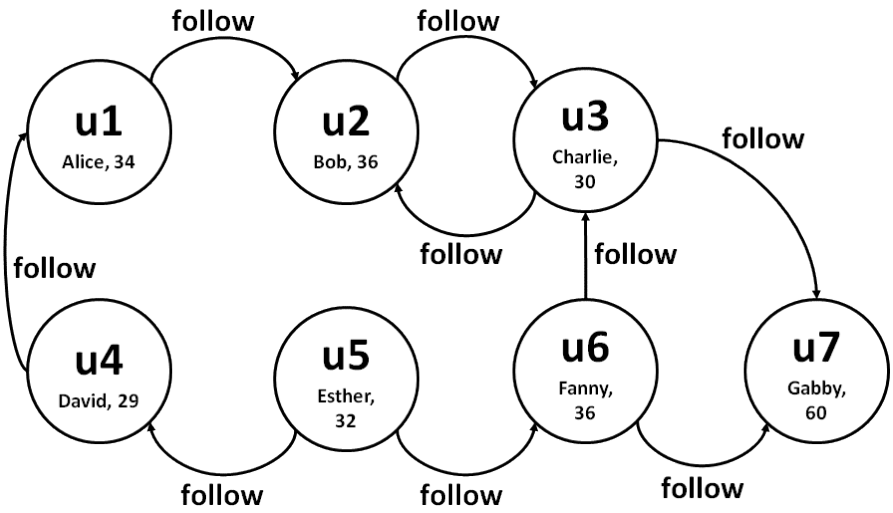
Last Name _____

The exam is **open book**

Part I

Answer the following questions. There is only one correct answer for each question.

1. (2 points) Consider the following graph and suppose g is its instantiation in GraphFrame.



Suppose the following command is executed on g:

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

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

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

v1	v2	v3
[u4, David, 29]	[u1, Alice, 34]	[u5, Esther, 32]

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

v1	v2	v3
[u5, Esther, 32]	[u4, David, 29]	[u6, Fanny, 36]

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

v1	v2	v3
[u2, Bob, 36]	[u3, Charlie, 30]	[u2, Bob, 36]

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

v1	v2	v3
[u3, Charlie, 30]	[u2, Bob, 36]	[u3, Charlie, 30]

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("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 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 {

                    // 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 512 MB.

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

Filename	Content
Cities1.txt	Beijing Cairo Delhi Dortmund Turin
Cities2.txt	Cairo Chongqing Delhi Istanbul Turin
Cities3.txt	Cairo Delhi Istanbul Milan 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 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)	1
part-r-00002 (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 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)	5
part-r-00001 (1 line)	5
part-r-00002 (1 line)	5

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 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)	7
part-r-00001 (1 line)	7
part-r-00002 (1 line)	7

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 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)	7
part-r-00001 (1 line)	0
part-r-00002 (1 line)	0

Part II

PoliWeather is an international company that monitors weather conditions around the globe. One of the key features tracked by the company is temperature. The management at PoliWeather is interested in calculating a set of statistics based on the following input data files that have been collected over the last sixty years.

- Sensors.txt
  - Sensors.txt is a textual file containing information about the sensors spread worldwide. There is one line for each sensor. The total number of sensors is greater than 10,000,000. This file is large. You cannot suppose the content of Sensors.txt can be stored in one in-memory Java/Python variable.
  - Each line of Sensors.txt has the following format
    - CodS, Latitude, Longitude, City, Country

      where *CodS* is the unique sensor identifier, *Latitude* and *Longitude* are its geolocation, while *City* and *Country* are the city and country the sensor is located in, respectively.
    - For example, the following line

      S1,40.7128° N,74.0060° W,New York City,USA

      means that the sensor with CodS ***S1*** is geolocated at (**40.7128° N, 74.0060° W**), which is in ***New York City***, ***USA***.

- Temperatures.txt
  - Temperatures.txt is a textual file containing information about the collected temperatures. Each line is associated with a temperature reading. This file is large. You cannot suppose the content of Temperatures.txt can be stored in one in-memory Java/Python variable.
  - Each line of Temperatures.txt has the following format
    - Timestamp, CodS, Temperature

      where *Timestamp* is the timestamp at which sensors with ID *CodS* have gathered the temperature. The gathered temperature value is stored in the *Temperature* field. The combination (Timestamp, CodS) uniquely identifies the readings, i.e., it is the primary key of this file.

      The timestamp format is YYYY/MM/DD-HH:MM.
    - For example, the following line

      2025/08/15-16:01,S2,36

      means that the temperature value recorded by sensor ***S2*** on **August 15, 2025**, at ***16:01*** was ***36*** degrees Celsius.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliWeather are interested in performing some analyses about the temperature values.

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

1. *Sensors with a minimum daily temperature below 0 °C on many dates.* Select the sensors that recorded a minimum daily temperature *below 0 °C on at least 300 dates.* Store the result in the output folder (one pair (CodS, Number of dates with a minimum daily temperature below 0 °C for Sensor CodS) for each of the selected sensors).

   Each output line has the following format:
   CodS, Number of dates with a minimum daily temperature below 0 °C for this Sensor CodS

Suppose that the input is Temperatures.txt and 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 three sensors (S1, S2, and S3).

Suppose that
- The Sensor S1 recorded a minimum daily temperature below 0 °C on 400 dates.
- The Sensor S2 recorded a minimum daily temperature below 0 °C on 153 dates.
- The Sensor S3 recorded a minimum daily temperature below 0 °C on 2010 dates.

The expected output for this toy example is as follows:

- S1,400
- S3,2010

- 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 PoliWeather have asked you to **develop a single Spark-based application** using either 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 Points 1 and 2, respectively).

1. *Cities where the maximum temperature in August 2025 exceeds that of August 2024.* The first part of this application considers only the temperature readings associated with August 2024 and August 2025. It selects the cities where the maximum temperature in August 2025 is higher than the maximum temperature recorded in the same city in August 2024. Store the result in the first output folder. The output contains one selected city per output line.

2. *Date(s) with the highest hourly temperature variation for each sensor*. The second part of this application considers all temperature readings. It computes the highest historical hourly temperature variation for each sensor and selects, for each sensor,

the date(s) associated with the calculated highest historical hourly temperature variation for each sensor.

Each hourly timeslot is defined as the combination (date, hour). For example, 2025/08/15-16 is the hourly timeslot associated with all the temperature readings collected from 16:00:00 to 16:59:59 on August 15, 2025, while 2025/08/16-16 is an example hourly timeslot associated with all the readings collected from 16:00:00 to 16:59:59 on August 16, 2025. Those are two different hourly timeslots (same hour but different date).

The hourly temperature variation recorded by a sensor on an hourly timeslot is defined as the difference between the maximum and minimum temperatures recorded by that sensor on that hourly timeslot.

The highest historical hourly temperature variation for a sensor is the highest hourly temperature variation calculated by considering all hourly variations associated with that sensor.

The result is stored in the second output folder (one distinct combination (CodS, Date) per output line for each date in which the highest historical hourly temperature variation was recorded by Sensor CodS).

Note that more than one date can be associated with the highest historical hourly temperature variation for each sensor. In that case, all the dates associated with the highest historical hourly temperature variation for each sensor must be selected (one combination (CodS, Date) per output line).
Suppose that the set of dates associated with the highest historical hourly temperature variation for each sensor can be large and cannot be stored in a Python set or list.

We remind you that the timestamp format is YYYY/MM/DD-HH:MM.

Part 2 - Example
Consider a toy example with only three sensors: S1, S2, and S3.

Suppose that
- The highest historical hourly temperature variation recorded by Sensor **S1** is **2.5** °C.
- The highest historical hourly temperature variation recorded by Sensor **S2** is **2.3** °C.
- The highest historical hourly temperature variation recorded by Sensor **S3** is **1.4** °C.

Suppose that
- Sensors S1 recorded the highest historical hourly temperature variation of 2.5 °C on the following hourly timeslot:
    - 2024/03/10-16 (i.e., March 10, 2024, from 16:00 to 16:59)

- Sensors S2 recorded the highest historical hourly temperature variation of 2.3 °C on the following hourly timeslots:
  - 2021/01/11-17 (i.e., January 11, 2021, from 17:00 to 17:59)
  - 2022/02/14-20 (i.e., February 14, 2022, from 20:00 to 20:59)
  - 2024/04/13-17 (i.e., April 13, 2024, from 17:00 to 17:59)

- Sensors S3 recorded the highest historical hourly temperature variation of 1.4 °C on the following hourly timeslots:
  - 2025/03/10-16 (i.e., March 10, 2025, from 16:00 to 16:59)
  - 2025/03/10-18 (i.e., March 10, 2025, from 18:00 to 18:59)
  - 2025/04/11-09 (i.e., April 11, 2025, from 09:00 to 09:59)

The content of the second output folder, given this toy example, is as follows:
S1,2024/03/10
S2,2021/01/11
S2,2022/02/14
S2,2024/04/13
S3,2025/03/10
S3,2025/04/11

- 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.