Big data processing and analytics

June 21, 2023

Student ID _____

First Name _____

Last Name _____

The exam is **open book**

Part I

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

1. (2 points) Consider the following Spark application.

```
package it.polito.bigdata.spark.exam;

import ....;
public class SparkDriver {

    public static void main(String[] args) {

        // Create a configuration object and set the name of the application
        SparkConf conf = new SparkConf().setAppName("Spark Code");

        // Create a Spark Context object
        JavaSparkContext sc = new JavaSparkContext(conf);

        JavaRDD<String> TempItalyRDD = sc.textFile("TemperatureItaly.txt");

        JavaRDD<String> TempFranceRDD = sc.textFile("TemperatureFrance.txt");

        // Compute the maximum value from TempItalyRDD
        double maxValueItaly = TempItalyRDD.reduce((a, b) -> max(a, b));

        // Compute the maximum value from TempFranceRDD
        double maxValueFrance = TempFranceRDD.reduce((a, b) -> max(a, b));

        // Create an RDD that contains the union of TempItalyRDD and TempFranceRDD.
        // Then, apply distinct and compute the maximum value
        JavaRDD<String> unionRDD = TempItalyRDD.union(TempFranceRDD);

        double maxValueUnion = unionRDD.distinct().reduce((a, b) -> max(a, b));

        // Print on the standard output the three maximum values
        System.out.println(maxValueItaly +" "+maxValueFrance +" "+maxValueUnion);

        // Store the content of unionRDD in the output folder
        unionRDD.saveAsTextFile("outputFolder/");
```

```
        // Close the Spark context
        sc.close();

    }

}
```

Suppose the input files TemperatureItaly.txt and TemperatureFrance.txt are read from HDFS. Suppose this Spark application is executed only once. Which one of the following statements is true?

a) This application reads the content of TemperatureItaly.txt 1 time and the content of TemperatureFrance.txt 1 times.

b) This application reads the content of TemperatureItaly.txt 2 times and the content of TemperatureFrance.txt 2 times.

c) This application reads the content of TemperatureItaly.txt 3 times and the content of TemperatureFrance.txt 3 times.

d) This application reads the content of TemperatureItaly.txt 4 times and the content of TemperatureFrance.txt 4 times.

2. (2 points) Consider the HDFS folder "inputFolder" containing the following two files:

Filename	Size	Content of the file
HumidityA.txt	16 bytes	51.45 9.55 8.15
HumidityB.txt	18 bytes	40.53 12.98 52.99

Suppose that you are using a Hadoop cluster that can potentially run up to 4 mappers in parallel and suppose that the HDFS block size is 2048MB.

Suppose that the following MapReduce program is executed by providing the folder "inputFolder" as input folder and the folder "results" as output folder. The number of reducers is set to 3.

```
/* Driver */
import … ;
public class DriverBigData extends Configured implements Tool {
        @Override
        public int run(String[] args) throws Exception {
                Configuration conf = this.getConf();
                Job job = Job.getInstance(conf);
                job.setJobName("Exam question");
                int N = Integer.parseInt(args[2]);

                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
                job.setJarByClass(DriverBigData.class);

                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                job.setMapperClass(MapperBigData.class);
                job.setMapOutputKeyClass(NullWritable.class);
                job.setMapOutputValueClass(DoubleWritable.class);

                job.setReducerClass(ReducerBigData.class);
                job.setOutputKeyClass(NullWritable.class);
                job.setOutputValueClass(DoubleWritable.class);

                job.setNumReduceTasks(N);

                if (job.waitForCompletion(true) == true)
                        return 0;
                else
                        return 1;
        }

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

/* Mapper */

```
import …;

class MapperBigData extends Mapper<LongWritable, Text, NullWritable,
DoubleWritable> {
        protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
                Double val = new Double(value.toString());
                context.write(NullWritable.get(), new DoubleWritable(val));
        }
}
```

/* Reducer */

```
import …;

class ReducerBigData extends Reducer<NullWritable, DoubleWritable,
NullWritable, DoubleWritable> {
        protected void reduce(NullWritable key, Iterable<DoubleWritable> values,
Context context) throws IOException, InterruptedException {

                double max = -1.0;
```

```
            for(DoubleWritable v : values) {
                if(v.get() > max)
                    max = v.get();
            }
            context.write(NullWritable.get(), new DoubleWritable(max));
        }
    }
```

What is the output generated by the execution of the application reported above?

a) Only one single part file containing 52.99

b) Two part files

- One containing the value 52.99
- One containing the value 51.45

c) Three part files

- One containing the value 52.99
- One containing the value 51.45
- One empty file

d) Three part files

- One containing the value 52.99
- Two empty files

Part II

PoliMeeting is an international company that manages online business meetings around the world. Statistics about the organized meetings and users are computed based on the following input data files, which have been collected in the company's latest 10 years of activity.

- Users.txt
  - Users.txt is a textual file containing the information about the users who participated to meetings organized by PoliMeeting. There is one line for each user and the total number of users is greater than 200,000,000. This file is large and you cannot suppose the content of Users.txt can be stored in one in-memory Java variable.
  - Each line of Users.txt has the following format
    - UID,Name,Surname,DateOfBirth,PricingPlan

      where *UID* is the user's unique identifier, *Name* and *Surname* are his/her name and surname, respectively, *DateOfBirth* is his/her date of birth, and

Plan is the type of pricing plan (free, business, etc.). The DateOfBirth format is YYYY/MM/DD.

- For example, the following line

  User1000,Mario,Rossi,1988/06/01,Business

  means that the name and surname of the user with identifier User1000 are Mario and Rossi, respectively, and that the customer was born on June 1, 1988. He has subscripted a Business pricing plan.

- Meetings.txt
  - Meetings.txt is a textual file containing the information about the events managed by PoliMeeting. There is one line for each meeting. The total number of meetings stored into Meetings.txt is greater than 1,000,000,000. This file is large and you cannot suppose the content of Meetings.txt can be stored in one in-memory Java variable.
  - Each line of Meetings.txt has the following format
    - MID,Title,StartTime,Duration,OrganizerUID

      where *MID* is the item unique identifier, *Title* is the title of the meeting, *StartTime* is the start time of the meeting, *Duration* is its duration in minutes, and *OrganizerUID* is the identifier of the user who organized the meeting.

      StartTime and EndTime are timestamps. The format of those two variables is YYYY/MM/DD-HH:MM:SS.

    - For example, the following line

      MID1034,Polito project kick-off,2023/02/07-20:40:00,90,User1000

      means that the meeting with MID ***MID1034*** was organized by User1000 and is titled ***"Polito project kick-off"***. It is scheduled for ***2023/02/07-20:40:00*** with a 90 minutes duration.

- Invitations.txt
  - Invitations.txt is a textual file containing information about invitations to meetings. A new line is inserted in Invitations.txt every time someone is invited to a meeting. Invitations.txt includes the historical data about the latest 10 years. This file is big and you cannot suppose the content of Purchases.txt can be stored in one in-memory Java variable.
  - Each line of Invitations.txt has the following format
    - MID,UID,Accepted

      where MID is the identifier of the meeting to which user UID has been invited. Accepted can assume three values: Yes, No, and Unknown, depending on the answer of the invited user.

    - For example, the following line

      MID1034,User1000,Yes

means that ***User1000*** has been invited to the meeting ***MID1034***, and he/she has accepted to participate.

Note that the same user can be invited to many meetings, and each meeting can have many invited users. Each combination (MID, UID) occurs at most one time in Invitations.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliMeeting are interested in performing some analyses about the largest meetings.

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

1. *The meeting with the largest number of expected participants*. The application selects the meeting with the largest number of expected participants, where the number of expected participants is the number of users who answered "Yes" to the invitation. If more than one meeting is associated with the largest number of participants, the one with the last MID, considering the alphabetical order, is selected. The MID of the selected meeting is stored in the output HDFS folder.

Suppose that the input is Invitations.txt and has been already set. Suppose that also the name of the output folder has been already 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 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"

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 PoliMeeting are interested in analyzing the characteristics of the meetings.

The managers of PoliMeeting asked you to develop one single application to address all the analyses they are interested in. The application has five arguments: the input files Users.txt, Meetings.txt, and Invitations.txt, and two output folders "outPart1/" and "outPart2/", which are associated with the outputs of the following points 1 and 2, respectively.

Specifically, design a single application, based on Spark, and write the corresponding code, to address the following two points:

1. *Statistics on the duration of the meetings considering only the meetings organized by users with a Business pricing plan.* The first part of this application considers only the users with a Business pricing plan (PrincingPlan="Business") who organized at least one meeting. For each user with a Business pricing plan who organized at least one meeting, compute (i) the average duration of the meetings he/she organized, (ii) the maximum duration of the meetings he/she organized, and (iii) the minimum duration of the meetings he/she organized. Store the result of this first part in the first HDFS output folder. Specifically, store one output line for each user with a Business pricing plan who organized at least one meeting. The format of each output line is as follows:

   UID, the average duration of the meetings organized by user UID, the maximum duration of the meetings organized by user UID, the minimum duration of the meetings organized by user UID

2. *Distribution of the number of invitations per meeting considering the users with a Business pricing plan.* Similarly to the first part, the second part of this application considers only the users with a Business pricing plan who organized at least one meeting. For each user with a Business pricing plan who organized at least one meeting, the second part of this application computes the distribution of the number of invitations per organized meeting. Specifically, calculate for each user with a Business pricing plan the number of large, medium, and small meetings he/she organized. A meeting associated with more than 20 invitations is classified as large. A meeting associated with a number of invitations between 20 and 5 is classified as medium. A meeting associated with fewer than 5 invitations is classified as small. For each user with a Business pricing plan who organized at least one meeting, store in the second HDFS the UID, the number of large meetings he/she organized, the number of medium meetings he/she organized, and the number of small meetings he/she organized (one output line per user).

Note. There are meetings without invitations. A meeting without invitations is considered a small meeting.

Example for the second part.

In this running example, suppose there are only two users with a Business pricing plan: UID1 and UID2.

Suppose that UID1 organized

- One meeting with 15 invitations → 1 medium meeting
- Two meeting with 16 invitations → 2 medium meetings
- One meeting with 25 invitations → 1 large meeting
- One meeting with 2 invitations → 1 small meeting

Suppose that UID2 organized

- Two meeting with 23 invitations → 2 large meeting
- One meeting with 4 invitations → 1 small meeting
- One meeting without invitations → 1 small meeting

The second output folder will contain the following two lines:

- UID1,1,3,1
- UID2,2,0,2

- You do not need to report imports. Focus on the content of the main method.
- Suppose both JavaSparkContext ***sc*** and SparkSession ***ss*** have been already set.
- 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"