Distributed architectures for big data processing and analytics

February 10, 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 application.

```
tempRDD = sc.textFile("Temperature.txt")


# Computes the number of lines of Temperature.txt
numLinesTemp = tempRDD.count()


# Select high temperature values
highTempRDD = tempRDD.map(lambda v: int(v))\
                     .filter(lambda v: v>30)

# Cache highTempRDD
highTempRDDCached = highTempRDD.cache()


# Computes the number of high temperatures
numHighTemp = highTempRDDCached.count()


# Store the content of highTempRDDCached in the output folder
highTempRDDCached.saveAsTextFile ("outputFolder/")


# Print on the standard output the computed values
print("Num lines: " + str(numLinesTemp))
print("Num high temperatures: " + str(numHighTemp))
```

Suppose the input file Temperature.txt is read from HDFS. Suppose this Spark application is executed only 1 time. Suppose highTempRDD is small enough to be completely cached into highTempRDDCached. Which one of the following statements is **true**?

a) This application reads the content of Temperature.txt 1 time.
b) This application reads the content of Temperature.txt 2 times.
c) This application reads the content of Temperature.txt 3 times.
d) This application reads the content of Temperature.txt 4 times.

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 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 …;

    // 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 {
            int sum = 0;


            // Consider only the keys starting with "D"
            if (key.toString().startsWith("D")) {
                    for (NullWritable value : values) {
                            sum++;
                    }

                    if (sum > 1) {
                            // Increment count
                            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 and Cities2.txt. Suppose the HDFS block size is 512 MB.

Content of Cities1.txt and Cities2.txt:

Filename (size and number of lines)	Content

Cities1.txt (80 bytes – 10 lines)	Beijing Cairo **Delhi** **Dhaka** **Dortmund** Mexico City Mumbai Mumbai Shanghai Tokyo
Cities2.txt (60 bytes – 7 lines)	Buenos Aires Chongqing **Delhi** **Dortmund** **Dortmund** Milan Mumbai

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 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 two part files is as follows.

Filename (number of lines)	Content
part-r-00000 (1 line)	2
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 two part files is as follows.

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

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 two 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 part-r-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-r-00000 (1 line)	3
part-r-00001 (0 line – empty file)	

Part II

PoliMeeting is an international company that manages online 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 15 years of activity.

- Users.txt
  - Users.txt is a textual file containing information about the users who organized or participated in meetings managed by PoliMeeting. There is one line for each user and the total number of users is greater than 150,000,000. This file is large and you cannot suppose the content of Users.txt can be stored in one in-memory variable.
  - Each line of Users.txt has the following format
    - UID,Name,Surname,Country,PricingPlan

      where *UID* 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 *PricingPlan* is the type of pricing plan (free, business, etc.).
    - For example, the following line

      User1000,Mario,Rossi,Italian,Business

means that the name and surname of the user with identifier User1000 are Mario and Rossi, respectively, and he is Italian. He has subscribed to a Business pricing plan.

- Meetings.txt
  - Meetings.txt is a textual file containing 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 2,000,000,000. This file is large and you cannot suppose the content of Meetings.txt can be stored in one in-memory variable.
  - Each line of Meetings.txt has the following format
    - MID,Title,StartTime,EndTime,OrganizerUID,MaxParticipants

      where *MID* is the item unique identifier, *Title* is the title of the meeting, *StartTime* is the start time of the meeting, *EndTime* is the end time of the meeting, *OrganizerUID* is the identifier of the user who organized the meeting and *MaxParticipants* is the maximum number of allowed participants.

      StartTime and EndTime are timestamps in the format YYYY/MM/DD-HH:MM:SS.
    - For example, the following line

      MID1034,Polito project kick-off,2024/02/07-20:30:00,2024/02/07-21:30:00,User1000,20

      means that the meeting with MID ***MID1034*** was organized by User1000, is titled ***"Polito project kick-off"***, and the maximum number of allowed participants is ***20***. The meeting was scheduled from ***2024/02/07-20:30:00*** to ***2024/02/07-21:30:00***.

- 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 15 years. This file is big and you cannot suppose the content of Purchases.txt can be stored in one in-memory 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 the invitation 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.

- Participations.txt
  - Participations.txt is a textual file containing information about who participated in the organized meetings. A new line is inserted in Participations.txt every time someone joins (participates in) a meeting. Participations.txt includes the historical data about the latest 15 years. This file is big and you cannot suppose the content of Participations.txt can be stored in one in-memory variable.
  - Each line of Participations.txt has the following format
    - MID,UID,JoinTimestamp,LeaveTimestamp
      where MID is the identifier of the meeting that user UID joined at *JoinTimestamp*. *LeaveTimestamp* is the timestamp at which UID left the meeting MID. The format of the timestamps *JoinTimestamp* and *LeaveTimestamp* is YYYY/MM/DD-HH:MM:SS.

    - For example, the following line
      MID1034,User10,2024/02/07-20:40:10, 2024/02/07-20:50:02

      means that ***User10*** joined the meeting ***MID1034*** on ***July 2, 2024***, at ***20:40:10*** and left it on ***July 2, 2024***, at ***20:50:02***.

    Note that the same user can participate in many meetings, and each meeting can have many participants. Moreover, **the same user can join and leave each meeting several times** (a new line associated with a different JoinTimestamp is inserted every time a user joins or rejoins the same meeting). Each triplet (MID, UID, JoinTimestamp) occurs at most one time in Participations.txt.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliMeeting are interested in performing some analyses about the pricing plans.

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

1. *Countries with many free and academic pricing plans*. The application selects the countries where at least 30% of their users have a free pricing plan (PrincingPlan='free') and at least 30% of their users have an academic pricing plan (PrincingPlan='academic'). The selected countries are stored in the output HDFS folder.

Output format (one line per each selected country):

country

Suppose that the input is Users.txt and has already been set. Suppose that also 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 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 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 PoliMeeting asked you to develop a single Spark-based application based either 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. *Users who frequently organized meetings with the maximum number of allowed participants in 2024.* The first part of this application selects the users who frequently organized meetings in 2024 (meetings with StartTime associated with 2024) with a number of actual participants equal to the maximum number of allowed participants (MaxParticipants). Specifically, a user is selected if more than 20 of the meetings the user organized in 2024 are characterized by a number of distinct actual participants equal to the maximum number of allowed participants. A user is considered an actual participant in a meeting if he/she participated in the meeting (according to the content

of Participations.txt). Store the identifiers (UIDs) of the selected users in the first HDFS output folder. Specifically, store one UID per output line.

Note. Remind that the same user can participate multiple times in the same meeting.

2. *Participation of the users in the meetings they organized in 2024*. The year of interest is again 2024. For each user who organized at least one meeting in the year 2024, compute the number of meetings he/she organized but did not participate. Store the result in the second HDFS output folder. Specifically, there is one output line for each user who organized at least one meeting. Each line contains the UID of one of the users who organized meetings, followed by the number of meetings UID organized in 2024 but did not participate in.

   Note. Those users who always participated in the meetings they organized must also be stored in the second output folder (for those users, the number of meetings they organized in 2024 but did not participate in is 0).

   Output format of each output line (second part):

   OrganizerUID, Number of meetings OrganizerUID organized in 2024 but did not participate in

   Example for the second part.

   In this small example, suppose there are only three users who organized meetings in 2024. The identifiers of these users are UID1, UID5, and UID12.

   Suppose that
   - UID1 organized **3** meetings in 2024 and participated in **2** of them.
   - UID5 organized **5** meetings in 2024 and participated in **5** of them.
   - UID12 organized **3** meetings in 2024 and participated in **0** of them.

   The second output folder must contain the following three lines:
   - UID1,1
   - UID5,0
   - UID12,3

- 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)