Big data processing and analytics

February 19, 2024

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("Exercise 19/02/2024 - Question 1");

    // Set the path of the input file/folder (if it is a folder, the job reads all the files in
    //the specified folder) for this job
    FileInputFormat.addInputPath(job, new Path("inputFolder/"));

    // Set the 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(IntWritable.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,
                           Text,
                           IntWritable,
                           NullWritable> {
```

```
	// Define count
	int count;


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

	protected void map(LongWritable key,
			Text value,
			Context context) throws IOException, InterruptedException {

		// Increment count
		count++;
	}



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

ReducerBigData.java

```
/* Reducer class */
package it.polito.bigdata.hadoop;
import …;

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

		// Emit the pair (key, NullWritable))
		context.write(key, 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 (size and number of lines)	Content
Cities1.txt (32 bytes – 4 lines)	Tokyo Delhi Shanghai São Paulo
Cities2.txt (34 bytes – 4 lines)	Tokyo Delhi New York City Karachi
Cities3.txt (18 bytes – 2 lines)	Mexico City Cairo

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 gen 29 14:01 part-r-00000

-rw-r--r-- 1 paolo paolo 2 gen 29 14:01 part-r-00001

-rw-r--r-- 1 paolo paolo 0 gen 29 14:01 part-r-00002

-rw-r--r-- 1 paolo paolo 0 gen 29 14:01 _SUCCESS

The content of the three part files is as follows.

Filename (number of lines)	Content
part-r-00000 (1 line)	4
part-r-00001 (1 line)	2
part-r-00002 (0 line – empty file)	

b) The content of the output folder is as follows.

-rw-r--r-- 1 paolo paolo 2 gen 29 14:01 part-r-00000

-rw-r--r-- 1 paolo paolo 2 gen 29 14:01 part-r-00001

-rw-r--r-- 1 paolo paolo 2 gen 29 14:01 part-r-00002

-rw-r--r-- 1 paolo paolo 0 gen 29 14:01 _SUCCESS

The content of the three part files is as follows.

Filename (number of lines)	Content
part-r-00000 (1 line)	4
part-r-00001 (1 line)	4
part-r-00002 (1 line)	2

c) The content of the output folder is as follows.

-rw-r--r-- 1 paolo paolo 4 gen 29 14:01 part-r-00000

-rw-r--r-- 1 paolo paolo 2 gen 29 14:01 part-r-00001

-rw-r--r-- 1 paolo paolo 0 gen 29 14:01 part-r-00002

-rw-r--r-- 1 paolo paolo 0 gen 29 14:01 _SUCCESS

The content of the three part files is as follows.

Filename (number of lines)	Content
part-r-00000 (2 lines)	4 4
part-r-00001 (1 line)	2
part-r-00002 (0 line – empty file)	

d) The content of the output folder is as follows.

-rw-r--r-- 1 paolo paolo 6 gen 29 14:01 part-r-00000

-rw-r--r-- 1 paolo paolo 0 gen 29 14:01 part-r-00001

-rw-r--r-- 1 paolo paolo 0 gen 29 14:01 part-r-00002

-rw-r--r-- 1 paolo paolo 0 gen 29 14:01 _SUCCESS

The content of the three part files is as follows.

Filename (number of lines)	Content
part-r-00000 (3 lines)	4 4 2
part-r-00001 (0 line – empty file)	
part-r-00002 (0 line – empty file)	

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

```
package it.polito.bigdata.spark;
import …;

public class SparkDriver {
        public static void main(String[] args) {
                SparkConf conf = new SparkConf().setAppName("Exam 24/02/05");
                JavaSparkContext sc = new JavaSparkContext(conf);


                JavaRDD<String> robotsRDD = sc.textFile("Robots.txt");


                // Compute the number of distinct cities in Robots.txt and
                // print it on stdout
                System.out.println("Distinct cities in Robot.txt: "+robotsRDD
                                                .map(line -> new String(line.split(",")[1]))
                                                .distinct()
                                                .count());

                JavaRDD<String> citiesRDD = sc.textFile("Cities.txt");

                // Print on the standard output the difference between the number of cities
                // and the number of distinct cities
                System.out.println("Diff: "+ citiesRDD.count()-citiesRDD.distinct().count());

                // Map the content of robotsRDD to (City, +1)
                JavaPairRDD<String, Integer> CityOneRobotRDD = robotsRDD
                                .mapToPair(line ->
                                        new Tuple2<String,    Integer>(line.split(",")[1], 1));

                // Map the content of citiesRDD to (City, +0)
                JavaPairRDD<String, Integer> AllCitiesRDD = citiesRDD
                                .mapToPair(line ->
                                        new Tuple2<String, Integer>(line.split(",")[0], 0));

                // Union CityOneRobotRDD and AllCitiesRDD
                JavaPairRDD<String, Integer> CityValuesRDD =
                                CityOneRobotRDD.union(AllCitiesRDD);

                // Compute the number of robots per city
                JavaPairRDD<String, Integer> CitiesNumRobotRDD =
                                 CityValuesRDD.reduceByKey((v1, v2) -> v1+v2);

                // Compute the minimum number of robots per city and print it on stdout
                int minRobots = CitiesNumRobotRDD.values()
                                .reduce((v1, v2) -> Math.min(v1, v2));
```

```
            System.out.println("Minimum number of robots per country: "+ minRobots);

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

Suppose the input files Robots.txt and Countries.txt are read from HDFS. Suppose this Spark application is executed **only 1 time**. Which one of the following statements is true?

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

Part II

PoliOnline is an international company that sells items online. To improve the sales and revenue of PoliOnline, a set of statistics about its items and users are computed based on the following input data sets/files.

- Catalogue.txt
  - Catalogue.txt is a textual file containing information about the items that are sold by PoliOnline. There is one line for each item and the total number of items is greater than 10,000,000. This file is large and you cannot suppose the content of Catalogue.txt can be stored in one in-memory Java variable.
  - Each line of Catalogue.txt has the following format
    - ItemID,Name,Category,StillinProduction

      where *ItemID* is the unique identifier of the item, *Name* is the name of ItemID, *Category* is its category (i.e., the item category), and *StillInProduction* is a string used to specify if ItemID is still in production or not (True/False).

    - For example, the following line

      ID1,t-shirt-winter,Clothing,True

      means that the item with ItemID ***ID1*** is characterized by the name ***t-shirt-winter***, it belongs to the ***Clothing*** category, and it is still in production (***True***).

- Users.txt
  - Users.txt is a textual file containing information about the customers/users who are registered on the PoliOnline website. There is one line for each user and the total number of users is greater than 100,000,000. This file is large and you cannot suppose the content of Customers.txt can be stored in one in-memory Java variable.
  - Each line of Users.txt has the following format
    - UserID,Name,Surname,City,Country

      where *UserID* is the unique identifier of the user, *Name* and *Surname* are his/her name and surname, respectively, and *City* and *Country* are the city and country where he/she lives.

    - For example, the following line

      User20,Paolo,Garza,Turin,Italy

      means that the name and surname of user ***User20*** are ***Paolo*** and ***Garza***, respectively, and that he lives in ***Turin*** (***Italy***).

- Purchases.txt
  - Purchases.txt is a textual file containing information about purchases. A new line is inserted in Purchases.txt every time an item is bought by a user (i.e., each line corresponds to one purchase). Purchases.txt contains historical data about the last 30 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 Purchases.txt has the following format
    - SaleTimestamp,UserID,ItemID,SalePrice

      where *SaleTimestamp* is the timestamp at which the user *UserID* bought the item identified by *ItemID*. *SalePrice* is the price of this purchase.

    - For example, the following line

      2019/02/02-09:15:01,User20,ID1,50.99

      means that on ***February 2, 2019***, at ***09:15:01*** the item identified by ***ID1*** was bought by user ***User20***, and ***User20*** bought that item for ***50.99*** euro. The format of SaleTimestamp is "YYYY/MM/DD-HH:MM:SS".

    Note that there is a many-to-many relationship among timestamps, users, and items (i.e., the triplet (SaleTimestamp, UserID, ItemID) is the "primary key" of Purchases.txt).

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of PoliOnline are interested in performing some analyses about users.

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

1. *Users who purchased many distinct items*. The application selects the users who purchased at least 50 distinct items in the period from 1/1/2020 to 31/12/2023. Store the identifiers of the selected users in the output HDFS folder (one UserID per output line). **Note**: There are users who have purchased millions of distinct items. The list of distinct items purchased by one single user is large and cannot be stored in a Java variable.

Suppose that the input is Purchases.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 and RDDs (19 points)

The managers of PoliOnline 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 Catalogue.txt, Users.txt, Purchases.txt, and two output folders (associated with the outputs of the following points 1 and 2, respectively).

1. *Users with the highest number of purchases in 2022 or 2023*. Considering only the purchases related to the years 2022 and 2023, the first part of this application aims to find the users associated with the highest number of purchases in the years 2022 or 2023. Specifically, a user is selected if (i) the number of purchases of that user in the year 2022 is equal to the maximum number of purchases in the year 2022 among all users or (ii) the number of purchases of that user in the year 2023 is equal to the maximum number of purchases in the year 2023 among all users. The first HDFS output folder must contain the identifiers of the selected users (one UserId per output line).
   Note: There is at least one purchase in the year 2022 and at least one purchase in the year 2023 (i.e., you do not have to deal with a maximum number of purchases equal to zero in this part of the problem).

   Example Part 1
   Only for this toy example, suppose there are only four users who purchased at least one item in the years 2022 or 2023: User1, User2, User3, and User4.
   Suppose that
   - User1 is associated with **55** purchases in the year 2022 and 10 purchases in the year 2023.User2 is associated with **55** purchases in the year 2022 and **100** purchases in the year 2023.
   - User3 is associated with 5 purchases in the year 2022 and **100** purchases in the year 2023.
   - User4 is associated with 10 purchases in the year 2022 and 10 purchases in the year 2023.

   In this case, the first part of the application stores the following content in the first output folder:

   - User1
   - User2
   - User3

2. *For each category, the items purchased by the largest amount of users in the last two years (2022-2023)*. Considering only the purchases related to 2022 and 2023, the second part of this application aims to find, for each category, the items purchased by the maximum number of unique users over the two years inside each category. If more than one item of the same category is associated with the maximum number of unique users for that category, select all those associated with the maximum value. Store the result in the second HDFS output folder (one pair (category, selected item) per output line). Output format: Category,ItemId. **Store the**

pair (Category, "NoPurchases") for the categories without purchases in the period 2022-2023.

Example Part 2
Only for this toy example, suppose there are only eight items:

- Item1 associated with the category Home and Kitchen
- Item2 associated with the category Home and Kitchen
- Item3 associated with the category Clothing
- Item4 associated with the category Clothing
- Item5 associated with the category Clothing
- Item6 associated with the category Fitness
- Item7 associated with the category Books
- Item8 associated with the category Books

Suppose that

- **1000** unique users purchased Item1 in the period 2022-2023
- 500 unique users purchased Item2 in the period 2022-2023
- **2000** unique users purchased Item3 in the period 2022-2023
- **2000** unique users purchased Item4 in the period 2022-2023
- 25 unique users purchased Item5 in the period 2022-2023
- 0 unique users purchased Item6 in the period 2022-2023, i.e., it was never purchased in the period 2022-2023
- **3000** unique users purchased Item7 in the period 2022-2023
- 20 unique users purchased Item8 in the period 2022-2023

Hence, in this case, it follows that:

- **Item1** is the item, among the items of the "**Home and Kitchen"** category, purchased by the maximum number of unique users.
- **Item3 and Item4** are the items, among the items of the "**Clothing"** category, purchased by the maximum number of unique users.
- **No items** belonging to the "**Fitness**" category have been purchased from 2022 to 2023.
- **Item7** is the item, among the items of the "**Books"** category, purchased by the maximum number of unique users.

In this case, the second part of the application stores the following content in the second output folder:

- Home and Kitchen,Item1
- Clothing,Item3
- Clothing,Item4
- Fitness,NoPurchases
- Books,Item7

- You do not need to write Java imports. Focus on the content of the main method.
- Suppose both **JavaSparkContext 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.
- 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".