Big data processing and analytics

June 12, 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.

```
# Read the input file containing student grades
# The format of each line is: StudentID,CourseID,Grade
gradesRDD = sc.textFile("Grades.txt")

# Count the total number of grade records in the file
totalGrades = gradesRDD.count()

# Filter for grades of the "Big Data" course (course ID "BIGD2025")
# and parse the grade as an integer.
bigDataGradesRDD = gradesRDD \
                .map(lambda line: line.split(',')) \
        .filter(lambda fields: fields[1] == "BIGD2025") \
        .map(lambda fields: int(fields[2]))

# Cache the RDD of Big Data grades
bigDataGradesCached = bigDataGradesRDD.cache()

# Calculate the number of students who took the Big Data exam
numStudentsBigData = bigDataGradesCached.count()

# Calculate the sum of all grades for the Big Data course
sumGradesBigData = bigDataGradesCached.reduce(lambda grade1, grade2: grade1 + grade2)
```

   Suppose the input file Grades.txt is read from HDFS and this Spark application is executed only **one** time. Suppose also that the content of bigDataGradesRDD is small enough to be completely stored in memory when cached.
   Which one of the following statements is **true**?

   a) The file Grades.txt is read from HDFS 1 time.
   b) The file Grades.txt is read from HDFS 2 times.
   c) The file Grades.txt is read from HDFS 3 times.
   d) The file Grades.txt is read from HDFS 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 1
    job.setNumReduceTasks(1);
```

```
    // 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 {


            // Input format: StudentID,CourseID,Grade
            String[] fields = value.toString().split(",");
            String courseID = fields[1];
            int grade = Integer.parseInt(fields[2]);
            context.write(new Text(courseID), new IntWritable(grade));

    }
}
```

ReducerBigData.java

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

    // Define count
    int count;
```

```
    protected void setup(Context context) {
    }

    protected void reduce(Text key, // Input key type
                          Iterable<NullWritable> values, // Input value type
                          Context context) throws IOException, InterruptedException {

        int passedStudentsCount = 0;
        for (IntWritable grade : values) {
            // A student is considered to have passed if the grade is >= 18
            if (grade.get() >= 18) {
                passedStudentsCount++;
            }
        }
        context.write(key, new IntWritable(passedStudentsCount)); }

    protected void cleanup(Context context) throws IOException, InterruptedException {
    }
}
```

Suppose that inputFolder contains the files Grades1.txt and Grades2.txt. Suppose the HDFS block size is 512 MB.

Filename (size and number of lines)	Content
Grades1.txt (4 lines)	s1,CS101,30 s2,CS101,17 s3,CS101,25 s4,DB202,28
Grades2.txt (4 lines)	s5,EL303,22 s6,EL303,16 s7,CS101,24 s8,DB202,18

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.

part-r-00000
part-r-00001
SUCCESS

The content of the two part files is as follows.

Filename	Content

part-r-00000	CS101 3 DB202 2 EL303 1
part-r-00001	

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

part-r-00000
part-r-00001
_SUCCESS

The content of the two part files is as follows.

Filename	Content
part-r-00000	CS101 3 DB202 2
part-r-00001	EL303 1

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

part-r-00000
_SUCCESS

The content of the part file is as follows.

Filename	Content
part-r-00000	CS101 3 DB202 2 EL303 1

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

part-r-00000
_SUCCESS

The content of the part file is as follows.

Filename	Content
part-r-00000	CS101 3 DB202 2

Part II

BigEnergy is an international company. BigEnergy collects historical energy consumption data from buildings and provides energy forecasts. It manages millions of buildings worldwide. Energy consumption data have been collected in tens of years of activity and are available in three files on an HDFS storage, which is structured as described in the following.

- **Buildings.csv**
  - This file contains descriptive information for each building. Each line in this file corresponds to a single building.
  - Format: **BuildingID,City,Country,SquareMeters,Type,Year,EnergyEfficiencyClass**
  - BuildingID: a unique identifier for the building (String).
  - City: the city where the building is located (String).
  - Country: the country where the building is located (String).
  - SquareMeters: the total floor area of the building in square meters (Integer).
  - Type: the type of building indicating its primary use, e.g., "Residential", "Commercial", "Industrial" (String).
  - Year: the year the building was constructed (Integer).
  - EnergyEfficiencyClass: the energy efficiency rating of the building, i.e., "A", "B", "C", “D”, “E”, “F”, “G” (String).
  - Example:
    - B_1001,Turin,Italy,120,Residential,1985,C
    - B_1002,Lyon,France,200,Commercial,2010,A
    - B_1003,Turin,Italy,50,Residential,1950,F

- **HistoricalEnergyConsumption.csv**
  - This file contains the historical hourly energy consumption for each building. **Each line represents the energy consumed by a specific building in a given hour and date**.
  - Format: **BuildingID,Timestamp,Energy_kWh**
  - BuildingID: the identifier of the building (String).
  - Timestamp: the timestamp of the measurement in the format YYYY/MM/DD-HH (String), using UTC for all buildings (to avoid local time DST transition issues). Each timestamp represents a specific hour in a specific date.
  - Energy_kWh: the energy consumed in that hour, measured in kilowatt-hours (Float).
  - The combination (BuildingID,Timestamp) uniquely identifies the lines of this file.
  - Example
    - B_1001,2024/03/15-14,5.2
    - B_1002,2024/03/15-14,25.8
    - B_1001,2024/03/15-15,5.5
    - B_1001,2024/03/15-16,5.7

- **EnergyConsumptionForecast.csv**

- This file contains the forecasted hourly energy consumption for each building. The structure is identical to that of the historical consumption file, but each line is a forecast.
- Format: **BuildingID,Timestamp,Forecast_kWh**
- BuildingID: the identifier of the building (String).
- Timestamp: the timestamp of the forecast in the format YYYY/MM/DD-HH (String), using UTC for all buildings. Each timestamp represents a specific hour in a specific date.
- Forecast_kWh: the forecasted energy consumption for that hour in kWh (Float).
- The combination (BuildingID,Timestamp) uniquely identifies the lines of this file.
- **Example**
  - B_1001,2024/03/15-14,5.1
  - B_1002,2024/03/15-14,26.5
  - B_1001,2024/03/15-15,5.3
  - B_1001,2024/03/15-16,5.7

Note that each building having a historical record for a given timestamp in HistoricalEnergyConsumption.csv, also has a corresponding forecast value for the same timestamp in EnergyConsumptionForecast.csv.

Note that each combination (BuildingID,Timestamp) occurs at most once in HistoricalEnergyConsumption.csv and at most once in EnergyConsumptionForecast.csv.

Exercise 1 – MapReduce and Hadoop (8 points)

Exercise 1.1

The managers of BigEnergy want to analyze the distribution of the energy efficiency classes of buildings based on their city and year of construction. Design a MapReduce application and write the corresponding Java code to address the following point:

- **Most frequent energy efficiency class by city and year of construction**. For each (city, year of construction) pair, the application must identify the most frequent energy efficiency class (i.e., the one that appears the highest number of times) among the buildings in that group. In case of a tie in the count, the energy class that comes first in alphabetical order must be chosen.

Each line in the output file must have the following format:

City,YearOfConstruction,MostFrequentEnergyClass

Note that the number of distinct values assumed by EnergyEfficiencyClass is seven, i.e., it is a small value.

Suppose that the input has already been set to Buildings.csv. 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 BigEnergy 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 Parts 1 and 2, respectively).

Part 1: Number of timestamps with forecast error above 20% for building type and energy efficiency class.
This task aims to analyze the performance of the energy forecast model. You are required to identify the absolute percentage difference between the historical and forecasted hourly energy consumption for each timestamp. Then, for each combination (energy efficiency class, type of building), count the total number of timestamps with an absolute percentage difference higher than 20%.
The absolute percentage difference for each timestamp is given by the following formula:
(abs(forecast – historical) / historical * 100)
Discard all records having zero historical hourly energy consumption.
Store in the output folder only the combinations (energy efficiency class, type of building) associated with at least one timestamp with an absolute percentage difference higher than 20%.

Output Format
Store the result in the first output folder. Each line must contain the energy efficiency class, the type of building, and the total number of timestamps with an absolute percentage difference higher than 20%.

- **Format**: EnergyEfficiencyClass,BuildingType,NumberOfTimestampsAbove20%

Part 2: Length of the peak energy consumption.
The task is to identify, for each year, the historical peak energy consumption per square meter, i.e., the overall maximum level of historical energy consumption per square meter across all buildings over the year. Then, for each building, compute the total number of timestamps spent at such peak consumption, for each year.
The energy consumption per square meter is computed by dividing the historical energy consumption associated with each timestamp provided in the "HistoricalEnergyConsumption.csv" by the square meters of the building provided in the "Buildings.csv".
Store in the output folder, for each year, only the buildings associated with the yearly peak consumption.

Output Format
Store the result in the second output folder. Each line of the output file must contain the year (e.g., 2024), the BuildingID, the value of the peak consumption per square meter (i.e., the highest energy consumption for that year), and the number of timestamps the building was at the peak consumption level.

- **Format**:
  Year,BuildingID,PeakConsumptionPerSquareMeter,NumberOfTimestampAtPeak
- E.g.,
  2024,B_1001,0.054,123
  2024,B_1002,0.054,21
  2023,B_1002,0.065,32

Toy example.

Suppose the context of HistoricalEnergyConsumption.csv is as follows.

B_1001,2024/03/15-14,10.0
B_1002,2024/03/15-14,5.0
B_1001,2024/03/15-15,5.5
B_1002,2024/03/15-15,4.3
B_1001,2024/03/15-16,5.5
B_1002,2024/03/15-16,4.3
B_1001,2025/03/15-14,15.0
B_1002,2025/03/15-14,5.8
B_1001,2025/03/15-15,15.0
B_1002,2025/03/15-15,6.2
B_1001,2025/03/15-15,14.0
B_1002,2025/03/15-15,6.0

Suppose the context of Buildings.csv is as follows.

B_1001,Turin,Italy,100,Residential,1985,C
B_1002,Lyon,France,50,Residential,2010,A

The peak energy consumption per square meter in 2024 is 0.1 (i.e., max(10.0/100, 5.0/50, 5.5/100, 4.3/50, 5.5/100, 4.3/50))

The peak energy consumption per square meter in 2025 is 0.15 (i.e., max(15.0/100, 5.8/50, 15.0/100, 6.2/50, 14.0/100, 6.0/50))

The second output folder, for this small toy example, contains

2024,B_1001,0.1,1
2024,B_1002,0.1,1
2025,B_1001,0.15,2

Note:

- 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)