# Big Data: Architectures and Data Analytics

June 26, 2018

Student ID _____

First Name _____

Last Name _____

## Part I

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

1. (2 points) Consider the HDFS folder "inputData" containing the following three files:

| Filename | Size | Content of the files | HDFS Blocks | |
|---|---|---|---|---|
| | | | Block ID | Content of the block |
| Prices1.txt | 18 bytes | 11.00<br>19.00<br>40.00 | B1 | 11.00<br>19.00 |
| | | | B2 | 40.00 |
| Prices2.txt | 18 bytes | 40.00<br>70.00<br>60.00 | B3 | 40.00<br>70.00 |
| | | | B4 | 60.00 |
| Prices3.txt | 18 bytes | 14.00<br>16.00<br>44.00 | B5 | 14.00<br>16.00 |
| | | | B6 | 44.00 |

Suppose that you are using a Hadoop cluster that can potentially run up to 10 mappers in parallel and suppose that the HDFS block size is 12 bytes.

Suppose that the following MapReduce program is executed by providing the folder "inputData" as input folder and the folder "results" as output folder.

```
/* 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("2018/06/26 - Theory");

                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(DoubleWritable.class);
                job.setMapOutputValueClass(NullWritable.class);

                job.setNumReduceTasks(0);

                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, DoubleWritable, NullWritable> {
        double sumPrices;

        protected void setup(Context context) {
                sumPrices = 0;
        }

        protected void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
                double price = Double.parseDouble(value.toString());
                sumPrices = sumPrices + price;
        }

        protected void cleanup(Context context) throws IOException, InterruptedException {
                // emit the content of sumPrices if it is greater than 50
                if (sumPrices>50)
                        context.write(new DoubleWritable(sumPrices), NullWritable.get());
        }
}
```

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

a) The output folder contains six part files

- One file that contains the following lines

  110

  60

- Five empty files

b) The output folder contains six part files

- One file that contains the following line

  30
- One file that contains the following line

  40
- One file that contains the following line

  110
- One file that contains the following line

  60
- One file that contains the following line

  30
- One file that contains the following line

  44

c) The output folder contains six part files

- One file that contains the following line

  110
- One file that contains the following line

  60
- Four empty files

d) The output folder contains six part files

- One file that contains the following line

  314
- Five empty files

2. (2 points) Consider the HDFS folder logsFolder, which contains three files: errors.txt, warning.txt, and log.txt. The size of errors.txt is 1036MB, the size of warning.txt is 500MB, while the size of log.txt is 1024MB. Suppose that you are using a Hadoop cluster that can potentially run up to 9 mappers in parallel and suppose to execute a Map-only MapReduce-based program that selects only the lines of the input files in logsFolder containing the words “ERROR” or “WARNING”.

The HDFS block size is 512MB. How many mappers are instantiated by Hadoop when you execute the application by specifying the folder logsFolder as input?

a) 3

b) 5

c) 6

d) 9

## Part II

PoliData is an international company managing large data centers around the world. To identify critical situations, PoliData computes a set of statistics about the managed virtual servers and their performances based on the following input data set/file.

- PerformanceLog.txt
  - PerformanceLog.txt is a text file containing the historical information about the percentage of resources used by the virtual servers managed by PoliData. For each virtual server, a new line is inserted in PerformanceLog.txt every 1 minute. The number of managed virtual servers is more than 100,000 and PerformanceLog.txt contains the historical data about the last year (i.e., PerformanceLog.txt contains more than 52 billion lines).
  - Each line of PerformanceLog.txt has the following format
    - Timestamp,VSID,CPUUsage%,RAMUsage%

      where *Timestamp* is the timestamp at which the Usage statistics are collected, *VSID* is the identifier of the monitored virtual server, *CPUUsage%* is the percentage of CPU Usage and *RAMUsage%* is the percentage of RAM Usage.

    - For example, the following line

      *2018/03/01,15:40,VS1,10.5,0.5*

      means that at ***15:40*** of ***March 1, 2018*** virtual server ***VS1*** was using ***10.5%*** of its CPU and ***0.5%*** of its RAM.

**Exercise 1 – MapReduce and Hadoop** (8 points)

The managers of PoliData are interested in selecting the virtual servers that were frequently characterized by a high CPU Usage during the Italian standard working hours (i.e., from 9:00 to 17:59) in May 2018.

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

A. *High CPU Usage during working hours*. Considering only the statistics (i.e., lines of PerformanceLog.txt) associated with the time slot from 9:00 to 17:59 of May 2018, the application must select the identifiers (VSID) of the virtual servers that were at least 10,000 times associated with a CPUUsage% value greater than 99.8. Store the result of the analysis in a HDFS folder. The output file contains one line for each of the selected virtual servers. Specifically, each line of the output file contains the VSID of one of the selected virtual servers.

The name of the output folder is an argument of the application. The other argument is the path of the input file PerformanceLog.txt. Note that PerformanceLog.txt contains the statistics of the last year but the analysis is focused only on May 2018 for the specific time slot from 9:00 to 17:59.

Fill out the provided template for the Driver of this exercise. Use your sheets of paper for the other parts (Mapper and Reducer).

**Exercise 2 – Spark and RDDs** (19 points)

The managers of PoliData are interested in performing some analyses about the average CPU and RAM Usage of each server during the last month (May 2018) to identify critical hours for each server. This analysis will be used to resize the resources allocated for some servers.

The managers of PoliData are also interested in identifying virtual servers characterized by an unbalanced daily usage of the CPU during the last month (May 2018). Specifically, given a date and a virtual server, the virtual server is characterized by an "*unbalanced daily usage of the CPU"* during that date if, for that server in that specific date, there are at least 8 hours for which the maximum value of CPUUsage% is greater than 90 and at least 8 hours for which the maximum value of CPUUsage% is less than 10.

The managers of PoliData asked you to develop one single application to address all the analyses they are interested in. The application has five arguments: CPU Usage threshold (*CPUthr*), RAM Usage threshold (*RAMthr*), the input file PerformanceLog.txt, and two output folders (associated with the outputs of the following points A and B, respectively).

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

A. (8 points) *Critical (VSID, hour) pairs in May 2018*. Considering only the statistics related to May 2018, the application must select only the pairs (VSID, hour) for which the average of CPUUsage% for that pair is greater than the specified threshold *CPUthr* and the average of RAMUsage% for that pair is greater than the specified threshold *RAMthr* (*CPUthr* and *RAMthr* are two arguments of the application). The application stores in the first HDFS output folder the information "VSID_hour" for the selected pairs (VSID, hour), one pair per line.

B. (11 points) *Pairs (VSID,date) characterized by a daily unbalanced usage of the CPU during May 2018*. Considering only the statistics related to May 2018, the application must select the pairs (VSID, date) that are characterized by an unbalanced daily usage of the CPU. Specifically, given a date and a virtual server, the virtual server is characterized by a daily unbalanced usage of the CPU during that date if, for that server in that specific date, there are at least 8 hours for which the maximum value of CPUUsage% is greater than 90 and at least 8 hours for which the maximum value of CPUUsage% is less than 10. The application stores in the second HDFS output folder the information "VSID_date" for the selected pairs (VSID, date), one pair per line.

# Big Data: Architectures and Data Analytics

June 26, 2018

Student ID _____

First Name _____

Last Name _____

## Use the following template for the Driver of Exercise 1

Fill in the missing parts. You can strikethrough the second job if you do not need it.

```
import ….
/* Driver class. */
public class DriverBigData extends Configured implements Tool {
        public int run(String[] args) throws Exception {
        Path inputPath = new Path(args[0]);
        Path outputDir = new Path(args[1]);
        Configuration conf = this.getConf();

        // First job
        Job job1 = Job.getInstance(conf);
        job1.setJobName("Exercise 1 - Job 1");
        // Job 1 - Input path
        FileInputFormat.addInputPath(job,_____);

        // Job 1 - Output path
        FileOutputFormat.setOutputPath(job,_____);

        // Job 1 - Driver class
        job1.setJarByClass(DriverBigData.class);

        // Job1 - Input format
        job1.setInputFormatClass(_____);

        // Job1 - Output format
        job1.setOutputFormatClass(_____);

        // Job 1 -  Mapper class
        job1.setMapperClass(Mapper1BigData.class);
        // Job 1 – Mapper: Output key and output value: data types/classes
        job1.setMapOutputKeyClass(_____);

        job1.setMapOutputValueClass(_____);

        // Job 1 -  Reducer class
        job.setReducerClass(Reducer1BigData.class);

        // Job 1 – Reducer: Output key and output value: data types/classes
        job1.setOutputKeyClass(_____);

        job1.setOutputValueClass(_____);

        // Job 1 - Number of reducers
        job1.setNumReduceTasks( 0[ _ ] or 1[ _ ] or >=1[ _ ] ); /* Select only one of the three options */
```

```
        // Execute the first job and wait for completion
        if (job1.waitForCompletion(true)==true)
        {
                // Second job
                Job job2 = Job.getInstance(conf);
                job2.setJobName("Exercise 1 - Job 2");
                // Set path of the input folder of the second job
                FileInputFormat.addInputPath(job2,_____);

                // Set path of the output folder for the second job
                FileOutputFormat.setOutputPath(job2,_____);

                // Class of the Driver for this job
                job2.setJarByClass(DriverBigData.class);

                // Set input format
                job2.setInputFormatClass(_____);

                // Set output format
                job2.setOutputFormatClass(_____);

                // Set map class
                job2.setMapperClass(Mapper2BigData.class);

                // Set map output key and value classes
                job2.setMapOutputKeyClass(_____);

                job2.setMapOutputValueClass(_____);

                // Set reduce class
                job2.setReducerClass(Reducer2BigData.class);

                // Set reduce output key and value classes
                job2.setOutputKeyClass(_____);

                job2.setOutputValueClass(_____);

                // Set number of reducers of the second job
                job2.setNumReduceTasks( 0[ _ ] or 1[ _ ] or >=1[ _ ] ); /*Select only one of the three
                                                                        options*/

                // Execute the job and wait for completion
                if (job2.waitForCompletion(true)==true)
                        exitCode=0;
                else
                        exitCode=1;
           }
           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);
        }
}
```