Big Data: Architectures and Data Analytics

July 18, 2019

Student ID ____

First Name ____

Last Name ____

The exam is **open book** and lasts **2 hours**.

Part I

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

1. (2 points) Consider an input HDFS folder *logsFolder* containing the files log2016.txt, log2017.txt, and log2018.txt. log2016.txt contains the logs of year 2016 and its size is 126MB, log2017.txt contains the logs of year 2017 and its size is 128MB, and log2018.txt contains the logs of year 2018 and its size is 258MB. Suppose that you are using a Hadoop cluster that can potentially run up to 20 instances of the mapper in parallel. Suppose to execute a MapReduce application that counts the number of lines containing the string 2017, by specifying *logsFolder* as input folder. The HDFS block size is 128MB. How many instances of the mapper are instantiated by Hadoop when you execute the application by specifying the folder *logsFolder* as input?

   a) 20

   b) 5

   c) 4

   d) 1

2. (2 points) Consider the HDFS folder “inputData” containing the following two files:

Filename	Size	Content of the file
Temperature1.txt	61 bytes	2016/01/01,00:00,0 2016/01/02,00:05,-1 2016/01/03,00:10,-1.2
Temperature2.txt	63 bytes	2016/01/01,00:15,-1.5 2016/01/01,00:20,0 2016/01/03,00:25,-0.5

Suppose that you are using a Hadoop cluster that can potentially run up to 10 instances of the mapper and 10 instances of the reducer in parallel and suppose that the HDFS block size is 512MB.

Suppose that the following MapReduce program, which computes the maximum temperature value for each input date, is executed by providing the folder “inputData” as input folder and the folder “results” as output folder.

```
/* Driver */
package it.polito.bigdata.hadoop.exam;

import ….;

public class DriverBigData extends Configured implements Tool {
        @Override
        public int run(String[] args) throws Exception {
                Path inputPath;
                Path outputDir;
                int exitCode;

                inputPath = new Path(args[0]);
                outputDir = new Path(args[1]);

                Configuration conf = this.getConf();

                Job job = Job.getInstance(conf);
                job.setJobName("Exercise #1 - Exam 2019/07/18");

                FileInputFormat.addInputPath(job, inputPath);
                FileOutputFormat.setOutputPath(job, outputDir);

                job.setJarByClass(DriverBigData.class);

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

                // Set mapper
                job.setMapperClass(MapperBigData.class);
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(DoubleWritable.class);

                // Set reduce class
                job.setReducerClass(ReducerBigData.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(DoubleWritable.class);

                // Set number of instances of the reducer class
                job.setNumReduceTasks(10);

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

                return exitCode;
        }
```

```
    public static void main(String args[]) throws Exception {
        // Exploit the ToolRunner class to "configure" and run the Hadoop
        // application
        int res = ToolRunner.run(new Configuration(), new DriverBigData(), args);

        System.exit(res);
    }

}
```

/* Mapper */

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

import java.io.IOException;

import ….;

/* Mapper */
class MapperBigData extends Mapper<LongWritable, Text, Text, DoubleWritable> {

    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String fields[] = value.toString().split(",");
        String date = fields[0];
        Double temperature = Double.parseDouble(fields[2]);

        // Emit (date, temperature)
        context.write(new Text(date), new DoubleWritable(temperature));
    }
}
```

/* Reducer */

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

import …..;

class ReducerBigData extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
    @Override
    protected void reduce(Text key, // Input key type
            Iterable<DoubleWritable> values, // Input value type
            Context context) throws IOException, InterruptedException {

        double maxTemp = Double.MIN_VALUE;

        // Iterate over the set of values and compute the maximum temperature
        for (DoubleWritable temperature : values) {
            if (temperature.get() > maxTemp) {
                maxTemp = temperature.get();
            }
        }

        // Emit (date, maximum temperature)
        context.write(key, new DoubleWritable(maxTemp));
    }
}
```

Suppose that the above MapReduce program is executed by providing the folder “inputData” as input folder. How many times the reduce method of this MapReduce program is invoked?

a) 2

b) 3

c) 6

d) 10

Part II

PoliSoftware is a non-profit company that monitors the quality of several software applications released around the world. Specifically, the analyses are based on the number of released patches. The computed statistics are based on the following input data set/file.

- Patches.txt
  - Patches.txt is a textual file containing the information about the patches of more than 10,000 software applications. It contains the data collected in the last 10 years (2010-2019).
  - Each line of Patches.txt has the following format
    - PID,Date,ApplicationName,BriefDescription

      where *PID* is the patch identifier, *Date* is the date in which the patch was released, *ApplicationName* is the name of the software application for which the patch was released, and *BriefDescrition* is a brief description of the patch.
    - For example, the following line

      PID7000,2017/10/01,Windows 10,Security patch

      means that the patch with id ***PID7000***, which is a patch for ***Windows 10***, was released on ***October 1, 2017*** and it is a ***Security Patch***.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliSoftware are interested in selecting the software applications characterized by an increasing number of patches in the years 2017-2018.

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

A. *Software applications with an increasing number of patches in the two years 2017-2018*. The application analyzes only the patches associated with years 2017 and 2018 and selects the software applications characterized by a number of patches in year 2018 greater than the number of patches released in year 2017. Store the selected applications in an HDFS folder. Each output line of the output contains one of the selected applications.

For instance, suppose that “LibreOffice 10” is associated with 10 patches in year 2017 and 15 patches in year 2018, while “Ubuntu 10.02” is characterized by 40 patches in year 2017 and 10 patches in year 2018. “LibreOffice 10” is selected and the string *LibreOffice 10* is stored in the output while “Ubuntu 10.02” is not selected.

The name of the output folder is an argument of the application. The other argument is the path of the input file Patches.txt, which contains the information about all the patches released in the last 10 years (2010-2019) but pay attention that the analysis we are interested in is focused only on years 2017 and 2018.

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 PoliSoftware are interested in performing some analyses about the patches of the analyzed software applications.

The managers of PoliSoftware asked you to develop one single application to address all the analyses they are interested in. The application has three arguments: the input file Patches.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) *Comparison between Windows 10 and Ubuntu 18.04 during year 2017*. Considering only the patches related in year 2017 and the applications “Windows 10” and “Ubuntu 18.04”, the Spark application selects for each month of year 2017 the application with more patches between “Windows 10” and “Ubuntu 18.04” (i.e., you must consider only the patches of those two applications and not the other patches). Specifically, for each month of year 2017, the Spark application
   - stores in the first HDFS output folder a line containing the pair (month, “W”) if in that month “Windows 10” is characterized by more patches than “Ubuntu 18.04”
   - stores in the first HDFS output folder a line containing the pair (month, “U”) if in that month “Ubuntu 18.04” is characterized by more patches than “Windows 10”
   - stores nothing in the first HDFS output folder for the months in which “Windows 10” and “Ubuntu 18.04” are characterized by the same number of patches

   For instance, suppose that
      - in January 2017 “Windows 10” is associated with 10 patches and “Ubuntu 18.04” is associated with 5 patches

- in February 2017 “Windows 10” is associated with 3 patches and “Ubuntu 18.04” is associated with 3 patches
- in March 2017 “Windows 10” is associated with 4 patches and “Ubuntu 18.04” is associated with 5 patches
- and in all the remaining months of year 2017 “Windows 10” and “Ubuntu 18.04” are characterized by the same number of patches

Then, the **content** of the **first output folder** will be
(1,”W”)
(3,”U”)

B. (11 points) *For each software application, compute the windows of three consecutive months with many patches in year 2018*. Considering all applications but only the patches of year 2018, the Spark application selects for each software application the windows of three consecutive months of year 2018 such that in each month of the window the software application is associated with at least 4 patches. The application stores in the **second HDFS output folder** all the selected windows (one window per line). Each output line contains the first month of one of the selected windows and the associated software application name (e.g., (1,”Acrobat”)).
Finally, the number of software applications associated with at least one of the selected windows is printed by the Spark application on the **standard output**.

For instance, suppose that

- “Acrobat” is associated with the following number of patches during year 2018:

Month	1	2	3	4	5	6	7	8	9	10	11	12
#Patches	**10**	**5**	**6**	1	**10**	**10**	**4**	**5**	1	**4**	**4**	**4**

- “Windows8” is associated with the following number of patches during year 2018:

Month	1	2	3	4	5	6	7	8	9	10	11	12
#Patches	**5**	**6**	**5**	**5**	0	1	0	**5**	1	3	**5**	**4**

- “LibreOffice1” is associated with the following number of patches during year 2018:

Month	1	2	3	4	5	6	7	8	9	10	11	12
#Patches	1	2	**6**	3	**4**	1	**5**	1	3	3	3	**4**

Then, the **content** of the **second output folder** will be
(1,“Acrobat”)
(5,“Acrobat”)
(6,“Acrobat”)
(10,“Acrobat”)
(1,“Windows8”)
(2,“Windows8”)

and **2** will be **printed on the standard output** because there are two software applications associated with the selected windows.

Big Data: Architectures and Data Analytics

July 18, 2019

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 instances of the reducer of the first Job
        job1.setNumReduceTasks( 0[ _ ] or exactly 1[ _ ] or any number >=1[ _ ] ); /* Select only one of
                                                                                     these 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(___________________________________);

        // Job 2 - Number of instances of the reducer of the second Job
        Job2.setNumReduceTasks( 0[ _ ] or exactly 1[ _ ] or any number >=1[ _ ] ); /* Select only
                                                                    one of these 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);
    }
}
```