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 1022MB, log2017.txt contains the logs of year 2017 and its size is 1024MB, and log2018.txt contains the logs of year 2018 and its size is 2050MB. Suppose that you are using a Hadoop cluster that can potentially run up to 15 instances of the mapper in parallel. Suppose to execute a MapReduce application that counts the number of lines containing the string 2018, by specifying *logsFolder* as input folder. The HDFS block size is 1024MB. How many instances of the mapper are instantiated by Hadoop when you execute the application by specifying the folder *logsFolder* as input?

   a) 3

   b) 4

   c) 5

   d) 15

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

Filename	Size	Content of the file
Temperature1.txt	61 bytes	2017/01/01,00:00,0 2017/01/02,00:05,-1 2017/01/03,00:10,-1.2
Temperature2.txt	63 bytes	2017/01/01,00:15,-1.5 2017/01/03,00:20,0 2018/01/05,00:25,-0.5

Suppose that you are using a Hadoop cluster that can potentially run up to 20 instances of the mapper and 20 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(20);

		// 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) 4

c) 6

d) 20

Part II

PoliGit is an international repository of large software projects. The managers of PoliGit are interested in analyzing the managed projects and the contributors. Specifically, the analyses are based on the number of project commits. The computed statistics are based on the following input data set/file.

- Commits.txt
  - Commits.txt is a textual file containing the information about the commits of more than 8,000 projects and more than 100,000 contributors. Each contributor can contribute to more than one project. Commits.txt contains the data collected in the last 20 years (2000-2019).
    Every time a contributor updates the content of some files of a project and commits those changes a new line is inserted in Commits.txt.
  - Each line of Commits.txt has the following format
    - CID,Timestamp,Project,ContributorNickname,Description

      where *CID* is the commit identifier, *Timestamp* is its timestamp, *Project* is the name of the updated/changed project, *ContributorNickname* is the unique nickname of the contributor who committed the changes associated with this commit operation, and *Description* is a brief description of the applied changes.
    - For example, the following line

      CID1201,2017/05/02_17:24,Apache Spark,JohnP,[SQL] Group By

      means that the commit operation with id ***CID1201***, which is a commit of a set of changes related to the ***Apache Spark*** project, was made on ***May 2, 2017*** at ***17:24*** by ***JohnP*** and it is associated with the “***[SQL] Group By***” issue.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliGit are interested in selecting the contributors with an increasing number of commits in the months May 2019-June 2019.

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

A. *Contributors with an increasing number of commits in the two months May 2019-June 2019*. The application analyzes only the commits associated with May 2019 and June 2019 and selects the contributors characterized by a number of commits in June 2019 greater than the number of commits made in May 2019. Store the selected contributors in an HDFS folder. Each output line of the output contains one contributor.

   For instance, suppose that JohnP is associated with 30 commits in May 2019 and 40 commits in June 2019, while PaoloP is associated with 5 commits in May 2019 and 2 commits in June 2019. JohnP is selected and the string *JohnP* is stored in the output while PaoloP 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 Commits.txt, which contains the information about all the commits made in the last 20 years (2000-2019) but pay attention that the analysis we are interested in is focused only on two months (May 2019 and June 2019).

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 PoliGit are interested in performing some analyses about the commits of the managed projects.

The managers of PoliGit asked you to develop one single application to address all the analyses they are interested in. The application has three arguments: the input file Commits.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 Apache Spark and Apache Flink during month June 2019*. Considering only the commits made in month June 2019 and the projects "Apache Spark" and "Apache Flink", the Spark application selects for each date of month June 2019 the project with more commits between "Apache Spark" and "Apache Flink" (i.e., you must consider only the commits of those two projects and not the other commits). Specifically, for each date of month June 2019, the Spark application
   - stores in the first HDFS output folder a line containing the pair (date, "AS") if in that date "Apache Spark" is characterized by more commits than "Apache Flink"
   - stores in the first HDFS output folder a line containing the pair (date, "AF") if in that date "Apache Flink" is characterized by more commits than "Apache Spark"
   - stores nothing in the first HDFS output folder for the dates in which "Apache Spark" and "Apache Flink" are characterized by the same number of commits

For instance, suppose that

- in June 1, 2019 "Apache Spark" is associated with 10 commits and "Apache Flink" is associated with 15 commits
- in June 2, 2019 "Apache Spark" is associated with 10 commits and "Apache Flink" is associated with 10 commits
- in June 3, 2019 "Apache Spark" is associated with 15 commits and "Apache Flink" is associated with 5 commits
- and in all the remaining dates of June 2019 "Apache Spark" and "Apache Flink" are characterized by the same number of commits

Then, the **content** of the **first output folder** will be

(2019/06/01,"AF")
(2019/06/03,"AS")

B. (11 points) *For each project, compute the windows of three consecutive months with many commits in year 2017*. Considering all projects but only the commits of year 2017, the Spark application selects for each project the windows of three consecutive months of year 2017 such that in each month of the window the project is associated with at least 20 commits. 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 project (e.g., (10,"Apache HBase")). Finally, the number of projects associated with at least one of the selected windows is printed by the Spark application on the **standard output**.

For instance, suppose that

- "Apache HBase" is associated with the following number of commits during year 2017:

Month	1	2	3	4	5	6	7	8	9	10	11	12
#Commits	**20**	**25**	**26**	10	**20**	**20**	**24**	**25**	18	**24**	**24**	**24**

- "Gimp" is associated with the following number of commits during year 2017:

Month	1	2	3	4	5	6	7	8	9	10	11	12
#Commits	**50**	**60**	15	**50**	0	**50**	**60**	**50**	**50**	19	**50**	**40**

- "LibreOffice1" is associated with the following number of commits during year 2017:

Month	1	2	3	4	5	6	7	8	9	10	11	12
#Commits	6	5	**60**	18	**40**	10	**50**	10	19	19	19	**40**

Then, the **content** of the **second output folder** will be

(1,"Apache HBase")
(5,"Apache HBase")
(6,"Apache HBase")
(10,"Apache HBase")
(6,"Gimp")
(7,"Gimp")

and **2** will be **printed on the standard output** because there are two projects 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);
        }
}
```