Distributed architectures for big data processing and analytics

Month Day, Year – Exam Example #5

Student ID ____

First Name ____

Last Name ____

The exam lasts **2 hours**

Part I

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

1. (2 points) Consider the following Spark application.

```
logsRDD = sc.textFile("logs.txt")

# Select URLs from logs
URLsRDD = logsRDD.map(lambda line: line.split(",")[0])

# Select distinct URLs
distinctURLs = URLsRDD.distinct()

# Count the number of distinct URLs
numDistinctURLs = distinctURLs.count()

print(numDistinctURLs)

# Select URLs starting with "A"
filteredLinesRDD = logsRDD.filter(lambda line: line.startswith("A"))

# Count the number of selected lines
numSelectedLines = filteredLinesRDD.count()

print(numSelectedLines)
```

Which one of the following statements is true?

a) The cache mechanism is useless in this application

b) Caching *logsRDD* can improve the efficiency of the application (in terms of execution time)

c) Caching *distinctURLs* can improve the efficiency of the application (in terms of execution time)

d) Caching *filteredLinesRDD* can improve the efficiency of the application (in terms of execution time)

2. (2 points) Consider an input HDFS folder *logsFolder* containing the files log2017.txt and log2018.txt. log2017.txt contains the logs of year 2017 and its size is 2052MB while log2018.txt contains the logs of year 2018 and its size is 252MB. Suppose that you are using a Hadoop cluster that can potentially run up to 10 instances of the mapper in parallel. Suppose to execute a map-only application, based on MapReduce, that selects the lines containing the string 2017, by specifying *logsFolder* as input folder. The HDFS block size is 256MB. How many mappers are instantiated by Hadoop when you execute the application by specifying the folder *logsFolder* as input?

   a) 11

   b) 10

   c) 9

   d) 1

Part II

PoliData is an international company managing large data centers around the world. Each managed server is equipped with a sensor that measures its inside temperature. Anomalous temperature values are stored and then analyzed by the administrators of the data centers. The statistics are computed by analyzing the following input data sets/files.

- Servers.txt
  - Servers.txt is a textual file containing the information about the servers managed by PoliData. There is one line for each server and the total number of servers is greater than 100,000 (i.e., Servers.txt contains more than 100,000 lines)
  - Each line of Servers.txt has the following format
    - SID,CPUVersion,DataCenterID,City,Country

      where *SID* is the server identifier, *CPUVersion* is the version of its CPU, *DataCenterID* is the identifier of the data center in which the server is located, *City* is the city in which the data center is located and *Country* is the country of *City.*

    - For example, the following line

      SID31,PentiumV,DC10,Turin,Italy

      means that the server with id ***SID31*** has a ***PentiumV*** CPU and it is located in the data center with id ***DC10***, which is located in ***Turin*** (***Italy***).

- Servers_TemperatureAnomalies.txt
  - Servers_TemperatureAnomalies.txt is a textual file containing the information about the anomalous temperatures of the servers managed by PoliData. It

contains the historical anomalies about the last 15 years. A new line is inserted in Servers_TemperatureAnomalies.txt every time an anomalous high temperature occurs inside a server.

- Each line of Servers_TemperatureAnomalies.txt has the following format
  - SID,Timestamp,AnomalousTemperatureValue

    where *Timestamp* is the timestamp at which an anomalous temperature was measured inside server SID, and *anomalousTemperatureValue* is the measured anomalous temperature value.
  - For example, the following line

    SID31,2018/03/01_15:40,90

    means that at ***15:40*** of ***March 1, 2018*** an anomalous temperature of 90°C was measured inside server ***SID31***.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliData are interested in analyzing the information about the CPU versions of the servers located in Spain.

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

A. *Spanish data centers with single CPU version*. The application analyzes the Spanish data centers (i.e., the data centers located in Spain). Specifically, a Spanish data center is selected if all its servers are characterized by the same CPU version. Store the identifiers of the selected data centers, and the associated CPU versions, in an HDFS folder. Each output line of the output contains one pair (DataCenterID,CPUVersion), one line per DataCenterID.

For example, suppose that all the servers of the data center with id DS31 are equipped with a PentiumV CPU while among the servers of data center DS12 some servers are equipped with a PentiumIV CPU and some with a 8086 CPU. DS31 is selected by the application and *DS31,PentiumV* is stored in the output folder while DS12 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 Servers.txt, which contains the information about all the servers used around the world but pay attention that the analysis we are interested in is focused only on the servers located in Spain.

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 anomalous temperatures measured inside the servers in the last nine years.

The managers of PoliData asked you to develop one single application to address all the analyses they are interested in. The application has four arguments: the two input files Servers.txt and Servers_TemperatureAnomalies.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 RDDs or Spark DataFrames, and write the corresponding code, to address the following points:

A. (9 points) *Critical servers during years 2010-2018*. Considering only the anomalies related to the last nine years (years 2010-2018) and only the anomalies with a measured temperature value greater than 100°C, the application selects the servers with more than fifty anomalies in at least one of the last nine years (i.e., a server is selected if there is at least one year (between 2010 and 2018) for which that server is associated with more than 50 anomalies with temperature>100°C). The application stores in the first HDFS output folder the SIDs of the selected servers (one SID per line).

B. (10 points) *Data centers characterized by servers with a limited number of anomalous temperatures during years 2010-2018.* Considering only the anomalies related to the last nine years (years 2010-2018), the application selects the identifiers of the data centers for which all servers are characterized by a limited number of temperature anomalies. Specifically, a data center is selected if all its servers are associated with at most 10 anomalies during years 2010-2018 (at most 10 anomalies for each server in the nine years). The application prints on the standard output the number of selected data centers and stores in the second HDFS output folder the identifiers of the selected data centers (one DataCenterID per line).

   For instance,

   - Suppose that in the data center DC31 there are three servers and suppose that during years 2010-2018 there were 6 anomalies for the first server, 5 anomalies for the second server, and 30 anomalies for the third server. DC31 **must not be** selected by the Spark application (Point B) because the third server is associated with more than 10 anomalies in the last nine years.

   - Suppose that in the data center DC10 there are two servers and suppose that during years 2010-2018 there were 6 anomalies for the first server and 7 anomalies for the second server. DC10 **must be** selected by the Spark application (Point B) because all its servers are associated with at most 10 anomalies in the last nine years.

   - Suppose that in the data center DC62 there are two servers and suppose that during years 2010-2018 there were 0 anomalies for the first server and 0 anomalies for the second server. DC62 **must be** selected by the Spark application (Point B) because all its servers are associated with at most 10 anomalies in the last nine years.

Note. Pay attention that there are servers without anomalies (i.e., some servers occur in *Servers.txt* but not in *Servers_TemperatureAnomalies.txt*).

Distributed architectures for big data processing and analytics

Month Day, Year – Exam Example #5

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);
        }
}
```