Big Data: Architectures and Data Analytics

July 16, 2018

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 the following driver of a Spark application.

```
package …
import ….
public class SparkDriver {
    public static void main(String[] args) {
        SparkConf conf=new SparkConf().setAppName("Exam");
        JavaSparkContext sc = new JavaSparkContext(conf);


        /* Analyze errors */
        JavaRDD<String> errorsRDD = sc.textFile("errors.txt");
        Long numErrors = errorsRDD.count();
        System.out.println(numErrors);


        /* Analyze warnings */
        JavaRDD<String> warningsRDD = sc.textFile("warnings.txt");
        JavaRDD<String> selWarningsRDD = warningsRDD.filter(w -> w.startsWith("warning 1"));

        Long numSelectedWarnings = selWarningsRDD.count();
        System.out.println(numSelectedWarnings);


        sc.close();
    }
}
```

Which one of the following statements is true?

a) Caching *errorsRDD* can improve the efficiency of the application (in terms of execution time)

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

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

d) Caching the RDDs of this Spark application does not improve its efficiency (in terms of execution time)

2. (2 points) Consider an input HDFS folder *inputFold* containing the files log1.txt and log2.txt. The size of log1.txt is 1024MB and the size of log2.txt is 256MB. Suppose that you are using a Hadoop cluster that can potentially run up to 5 instances of the mapper in parallel and suppose to execute the word count application, based on MapReduce, by specifying *inputFold* as input folder. Which of the following values is a proper HDFS block size if you want to "force" Hadoop to run 5 instances of the mapper in parallel when you execute the word count application by specifying *inputFold* as input folder?

a) Block size: 256MB

b) Block size: 512MB

c) Block size: 1024MB

d) Block size: 2048MB

Part II

PoliData is an international company managing data centers around the world. To identify critical servers, PoliData computes a set of statistics about failures and downtime minutes of its servers based on the following input data sets/files.

- Servers.txt
    - Servers.txt is a text file containing the list of servers managed by PoliData. Servers.txt contains one line for each server. The number of managed servers is more than 100,000.
    - Each line of Servers.txt has the following format
        - SID,IP,DataCenterID

          where *SID* is the server identifier, *IP* is its IP address, and *DataCenterID* is the identifier of the data center in which the server is placed.
        - For example, the following line

          S1,130.192.12.21,DC10

          means that the IP address of server ***S1*** is ***130.192.12.21*** and server ***S1*** is placed in data center ***DC10***.
- Failures.txt
    - Failures.txt is a text file containing the historical information about the failures of the managed servers. A new line is inserted in Failures.txt every time a failure occurs. The number of managed servers is more than 100,000 and Failures.txt contains the historical data about the failures of the last 10 years.
    - Each line of Failures.txt has the following format
        - Date,Time,SID,FailureType,Downtime

          where *Date* is the date of the failure, *Time* is the time when the failure was detected, *SID* is the identifier of the server associated with this failure, *FailureType* is the type of failure, and *Downtime* is the number of minutes for which the server was out of service due to its failure.
        - For example, the following line

          2017/05/01,05:40:31,S1,hard_drive,15

          means that at ***05:40:31*** (hour=05, minute=40, second=31) of ***May 1, 2017*** a ***hard drive*** failure was detected affecting server ***S1*** and ***S1*** was out of service for ***15*** minutes due to that failure.

Exercise 1 – MapReduce and Hadoop (8 points)

The managers of PoliData are interested in selecting the servers that were affected at least one time by a hard drive failure and at least one time by a RAM failure in April 2016.

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

A. *SIDs of the servers with hard drive and RAM failures in April 2016*. Considering only the failures detected in April 2016, the application must select the identifiers (SIDs) of the servers that were affected **by both types of failures** in **April 2016:** at least one time by a **hard drive** failure in April 2016 **AND** at least one time by a **RAM** failure in April 2016. Store the result of the analysis in a HDFS folder. The output contains one SID per line.

The arguments of the Hadoop application are (i) the path of the input file Failures.txt and (ii) the name of the output folder. Note that Failures.txt contains the failures about the last 10 years but the analysis we are interested in is limited to April 2016 only.

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 (A) performing some analyses about the number of failures for each data center in year 2017 and (B) identifying “faulty servers in 2017”.

The managers of PoliData asked you to develop one single application to address all the analyses they are interested in. The application has the following arguments: the path of the input file Servers.txt, the path of the input file Failures.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) Number of *failures per data center in year 2017*. Considering only the failures related to year 2017, the application must select the identifiers of the data centers (DataCenterIDs) affected by at least 365 failures during year 2017 and store in the first HDFS output folder the information “DataCenterID,Number of failures in year 2017” for the selected data centers (one data center per line).

B. (11 points) *SIDs of “Faulty servers in 2017”*. Considering only the failures related to year 2017, the application must select the identifiers (SIDs) of the servers classified as “faulty server in 2017”. A server is classified as a “faulty server in 2017” if (i) it was affected by **at least 2 failures in each of the 12 months of year 2017** and (ii) its **total downtime was more than 1440 minutes during year 2017** (i.e., the sum of the values of Downtime for that server during year 2017 is greater than 1440 minutes). The application stores in the second HDFS output folder the information about the identifiers (SIDs) of the selected servers (one SID per line).

Big Data: Architectures and Data Analytics

July 16, 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 instances of the reducer of the first job
        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(___________________________________);

        // Job 2 - Number instances of the reducer 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);
    }
}
```