Distributed architectures for big data processing and analytics

June 27, 2020

Student ID _____

First Name _____

Last Name _____

The exam lasts **90 minutes**

Part I

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

1. (2 points) Consider the input HDFS folder *failuresFolder* that contains the following two files:

   - failures2017.txt: size=1027MB
   - failures2018.txt: size=1028MB

   Suppose that you are using a Hadoop cluster that can potentially run up to 10 instances of the mapper class in parallel. Suppose to execute a MapReduce application for Hadoop that computes some statistics about the failures. This application is based on one single MapReduce job. The input of this Hadoop application is the HDFS folder *failuresFolder*. The HDFS block size is 1024MB. The number of instances of the reducer class is set to 5. How many mappers are instantiated by Hadoop (i.e., how many instances of the mapper class) when you execute this MapReduce application by specifying the folder *failuresFolder* as input?

   a) 5

   b) 4

   c) 3

   d) 2

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

```
# Define an accumulator.
# Initialize it to 0
invalidEmails = sc.accumulator(0)

# Create in RDD from a local python list
emailsRDD= sc.parallelize(["paolo.garza@polito.it",
```

```
                                    "p.garza@polito.it",
                                    "p.garza_polito.it",
                                    "test_polito.it",
                                     "test@polito.it"])

# Filtering function
def validEmailFunc(line):
    if (line.find('@')<0):
        # This email does not contain the symbol @.
        # Increment the number of invalid emails and return False
        invalidEmails.add(1)
        return False
    else:
        # This email contains the symbol @. Valid email. Return True.
        return True

# Apply the filter that selects only the input emails that contain the @ symbol
validEmailsRDD= emailsRDD.filter(validEmailFunc)

# Print the number of invalid emails
print("Invalid emails: ", invalidEmails.value)

# Count the number of valid emails and print it on the standard output of the driver
print("Valid emails: ", validEmailsRDD.count() )

# Store the valid emails in the output folder "outFolder"
validEmailsRDD.saveAsTextFile("outFolder")
```

Which one of the following statements is true?

a) The execution of this application prints on the standard output of the driver the following result

   Invalid emails: 0
   Valid emails: 3

b) The execution of this application prints on the standard output of the driver the following result

   Invalid emails: 2
   Valid emails: 3

c) The execution of this application prints on the standard output of the driver the following result

   Invalid emails: 4
   Valid emails: 3

d) Accumulators cannot be incremented in the function associated with an RDD filter transformation.

Part II

PoliCars is an international car sharing company. It has cars in over 1000 cities around the world. In each city, it has hundreds of cars. PoliCars computes a set of statistics to characterize and identify frequent failures of its cars. The analyses are performed by considering the following input data sets/files.

- Cars.txt
  - Cars.txt is a text file containing the list of cars managed by PoliCars. One line for each car of PoliCars is stored in Cars.txt. The number of cars is more than 40000.
  - Each line of Cars.txt has the following format
    - CarID,Model,Company,City

      where *CarID* is the car identifier, *Model* is its model, *Company* is the name of its carmaker company, and *City* is the city in which that car is used.

    - For example, the following line

      Car12,Panda,FCA,Paris

      means that car ***Car12*** is used in ***Paris***. ***Car12*** is a ***Panda*** (i.e., Panda is the model of Car12) and it was manufactured by ***FCA***.

- CarsFailures.txt
  - CarsFailures.txt is a text file. It stores the historical failures of the cars. A new line is inserted in CarsFailures.txt every time a new failure occurs. CarsFailures.txt contains the historical data about the failures of the last 30 years.
  - Each line of CarsFailures.txt has the following format
    - Date,Time,CarID,FailureType

      where *CarID* is the identifier of the car that had a failure of type *FailureType* at time *Time* of the date *Date*.

    - For example, the following line

      2015/01/05,08:45,Car15,Engine

      means that car ***Car15*** had a failure of type ***Engine*** at ***08:45*** (hour=08, minute=45) of ***January 5, 2015***.

Exercise 1 – MapReduce and Hadoop (7 points)

The managers of PoliCars are interested in performing some analyses about the failures of their cars.

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

1. *Cars with frequent and different types of failures in year 2018*. The application considers only the failures of years 2018 and selects the CarIDs of the cars that had at least 5 failures during year 2018 and at least two different types of failures in year 2018. Store the identifiers (CarIDs) of the selected cars, and the associated number of failures in year 2018, in the output HDFS folder. Each output line contains one pair (CarIDs,NumberOfFailuresYear2018), one line per selected car.

Suppose that the input is CarsFailures.txt and the name of the output folder has been already set.

- Write only the content of the Mapper and Reducer classes (map and reduce methods. setup and cleanup if needed). The content of the Driver must not be reported.

- Use the next two specific multiple-choice questions to specify the number of instances of the reducer class for each job.

- If your application is based on two jobs, specify which methods are associated with the first job and which are associated with the second job.

- If you need personalized classes report for each of them:

    - name of the class

    - attributes/fields of the class (data type and name)

    - personalized methods (if any), e.g, the content of the toString() method if you override it

    - do not report get and set methods. I suppose they are "automatically defined"

Exercise 1 - Number of instances of the reducer - Job 1 - MapReduce and Hadoop (0.5 points)

Select the number of instances of the reducer class of the first Job

(a) 0

(b) exactly 1

(c) any number >=1

Exercise 1 - Number of instances of the reducer - Job 2 - MapReduce and Hadoop (0.5 points)

Select the number of instances of the reducer class of the second Job

(a) One single job is needed for this MapReduce application

(b) 0

(c) exactly 1

(d) any number >=1

Exercise 2 – Spark and RDDs (19 points)

The managers of PoliCars are interested in performing some analyses about the failures of their cars.

The managers of PoliCars 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 Cars.txt and CarsFailures.txt and two output folders, "outPart1/" and "outPart2/", which are associated with the outputs of the following Points 1 and 2, respectively.

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

1. *Cars with an increasing number of failures in 2018.* The application selects the cars with a number of engine failures (FailureType='Engine') in year 2018 that is greater than the number of engine failures in year 2017 (i.e., a car is selected if the number of engine failures for that car in year 2018 is greater than the number of engine failures for the same car in year 2017). The application stores in the first HDFS output folder the CarIDs of the selected cars and their models (one pair (CarID,Model) per line).

2. *Failures in consecutive dates*. The application must select, for each car, all the sequences of two consecutive dates with at least one failure per date (consider all the historical data available in CarsFailures.txt). Specifically, given a car and a sequence of two consecutive dates, that sequence of two consecutive dates is selected for that car if and only if in both dates that car had at least one failure. The application stores the result in the second HDFS output folder. Specifically, each of the selected combinations (car, sequence of two consecutive dates with at least one failure per date) is stored in one output line and the used format is the following: CarID, first date of the selected sequence of two consecutive dates with at least one failure per date.

   Note. Pay attention that each **car can have more than one failure in the same date**.

   You can have overlapped sequences of two consecutive dates among the selected sequences.

   Suppose that someone has already implemented the following Python function

   - *previousDate(String date)*

- The parameter of this function is a string representing a date (in the format yyyy/mm/dd). The returned value is a string representing the previous date

- For example, the invocation

  yesterday=previousDate("2015/05/02")

  returns "2015/05/01" and stores "2015/05/01" in the variable *yesterday.*

Point 2: Examples

For instance, suppose that car Car12 had at least one failure in May 2, 2015 and at least one failure in May 3, 2015. It means that this sequence of two consecutive dates **must be selected** for Car12 and the following line is stored in the output folder: Car12,2015/05/02.

For instance, suppose that car Car15 had three failures in May 2, 2015 and no failures in May 3, 2015. It means that this sequence of two consecutive dates **must not be selected** for Car15.

You can have overlapped sequences of two consecutive dates among the selected sequences. For instance, suppose that car Car10 had at least one failure in May 2, 2015, at least one failure in May 3, 2015, and at least one failure in May 4, 2015. It means that in this case **two overlapped sequences** must be selected for Car10. Specifically, **two output lines** are stored in the output folder in this case:

- Car10,2015/05/02
- Car10,2015/05/03

The first output line is associated with the first sequence (2015/05/02, 2015/05/03) in which Car10 had at least one failure per date while the second output line is associated with the second sequence (2015/05/03, 2015/05/04) in which Car10 had at least one failure per date.

Suppose sc (Spark Context) and spark (Spark Session) have been already set.