# Distributed architectures for big data processing and analytics

June 21, 2021

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 an HDFS folder containing the files dataTurin.txt and dataRome.txt. The size of dataTurin.txt is 1027MB, the size of dataRome.txt is 1021MB. How many blocks are used to store those two files in HDFS if the HDFS block size is 2048MB? Suppose the replication factor is set to 1 (i.e., no replication).

   a) 1 block

   b) 2 blocks

   c) 3 blocks

   d) 6 blocks

2. (2 points) Consider the following Spark Streaming applications.

**(Application A)**

```
from pyspark.streaming import StreamingContext

# Create a Spark Streaming Context object
ssc = StreamingContext(sc, 10)

# Create a (Receiver) DStream that will connect to localhost:9999
# and define windows
inputWindowDStream = ssc.socketTextStream("localhost", 9999)\
.window(20, 10)\
.map(lambda value: int(value))

#Apply a filter
filteredDStream = inputWindowDStream.filter(lambda value : value>5)

# Compute the maximum value
resDStream = filteredDStream.reduce(lambda v1,v2:max( v1,v2))
```

```
# Print the result
resDStream.pprint()

ssc.start()
ssc.awaitTerminationOrTimeout(360)
ssc.stop(stopSparkContext=False)
```

**(Application B)**

```
from pyspark.streaming import StreamingContext

# Create a Spark Streaming Context object
ssc = StreamingContext(sc, 10)

# Create a (Receiver) DStream that will connect to localhost:9999
inputDStream = ssc.socketTextStream("localhost", 9999)\
.map(lambda value: int(value))

#Apply a filter
# and define windows
filteredDStream = inputDStream.filter(lambda value : value>5)\
.window(20, 10)


# Compute the maximum value
resDStream = filteredDStream.reduce(lambda v1,v2:max( v1,v2))

# Print the result
resDStream.pprint()

ssc.start()
ssc.awaitTerminationOrTimeout(360)
ssc.stop(stopSparkContext=False)
```

**(Application C)**

```
from pyspark.streaming import StreamingContext

# Create a Spark Streaming Context object
ssc = StreamingContext(sc, 10)

# Create a (Receiver) DStream that will connect to localhost:9999
# and define windows
inputWindowDStream = ssc.socketTextStream("localhost", 9999)\
.map(lambda value: int(value))\
.window(20, 10)


#Apply a filter
filteredDStream = inputWindowDStream.filter(lambda value : value>5)

# Compute the maximum value
resDStream = filteredDStream.reduce(lambda v1,v2:max( v1,v2))

# Print the result
resDStream.pprint()
```

if we have window after the reduce which is an aggregation function, then we would have two max values related to two batches of the window

```
ssc.start()
ssc.awaitTerminationOrTimeout(360)
ssc.stop(stopSparkContext=False)
```

Which one of the following statements is true?

a) Applications A, B, and C are equivalent in terms of returned result, i.e., given the same input they return the same result.

b) Applications A and B are equivalent in terms of returned result, i.e., given the same input they return the same result, while C is not equivalent to the other two applications.

c) Applications A and C are equivalent in terms of returned result, i.e., given the same input they return the same result, while B is not equivalent to the other two applications.

d) Applications A, B, and C are different in terms of returned result, i.e., given the same input they may return different results (the result of A is different from the result of B and the result of A is different from the result of C and the result of B is different from the result of C).

# Part II

PoliCars is an international association that monitors car sales around the world. PoliCars tracks all car sales independently of the manufacturer. PoliCars computes a set of statistics about car sales. The analyses are performed by considering the following input data sets/files.

- CarModels.txt
  - CarModels.txt is a text file containing the list of car models produced around the world. Each line of CarModels.txt is associated with one car model. The number of distinct car models is large and you cannot suppose the content of CarModels.txt can be stored in one in-memory python variable.
  - Each line of CarModels.txt has the following format
    - ModelID,ModelName,Manufacturer

      where *ModelID* is the unique car model identifier while *ModelName* and *Manufacturer* are the name of the car model and the manufacturer of the car model, respectively.

    - For example, the following line

      *Model10,PandaPop,FIAT*

      means that the name of the car model identified by ***Model10*** is ***PandaPop*** and its manufacturer is ***FIAT***.

- SalesEU.txt
  - SalesEU.txt is a text file containing the information about the sales in Europe. One line for each European sale is stored in SalesEU.txt. This file contains all sales of the last years in Europe (more than 50 years of sales). This file is large and you cannot suppose the content of SalesEU.txt can be stored in one in-memory python variable.
  - Each line of SalesEU.txt has the following format
    - SID,CarID,ModelID,Date,Country,Price

      where *SID* is the unique identifier of the sale while *CarID* is the identifier of the car that was sold and *ModelID* is its model, *Date* is the date of sale, *Country* is the country where the car was sold and *Price* is the sale price. The Date format is “YYYY/MM/DD”. Note that the same car can be sold multiple times. Hence, the same CarID can occur in different lines associated with different sales.

    - For example, the following line

      *SIDEU10,CarIDFT2011,Model10,1985/05/02,Italy,5900*

means that the sale identified by ***SIDEU10*** is related to the sale of the car identified by the code ***CarIDFT2011***. Car ***CarIDFT2011*** was sold on ***May 2, 1985*** in ***Italy*** at ***5900*** euro. The model of the car is ***Model10***.

- SalesExtraEU.txt
  - SalesExtraEU.txt is a text file containing the information on the sales outside Europe. One line for each non-European sale is stored in SalesExtraEU.txt. This file contains all sales of the last years made outside Europe (more than 50 years of sales). This file is large and you cannot suppose the content of SalesExtraEU.txt can be stored in one in-memory python variable.
  - SalesExtraEU.txt has the same format of SalesEU.txt. In particular, each line of SalesExtraEU.txt has the following format
    - SID,CarID,ModelID,Date,Country,Price

      where *SID* is the unique identifier of the sale while *CarID* is the identifier of the car that was sold and *ModelID* is its model, *Date* is the date of sale, *Country* is the country where the car was sold and *Price* is the sale price. The Date format is "YYYY/MM/DD". Note that the same car can be sold multiple times. Hence, the same CarID can occur in different lines associated with different sales.

    - For example, the following line

      *SIDExtraEU15,CarIDFS2041,Model10,1995/04/01,Argentina,2500*

      means that the sale identified by ***SIDExtraEU15*** is related to the sale of the car identified by the code ***CarIDFS2041***. Car ***CarIDFS2041*** was sold on ***April 1, 1995*** in ***Argentina*** at ***2500*** euro. The model of the car is ***Model10***.

**Exercise 1 – MapReduce and Hadoop** (7 points)

The managers of PoliCars are interested in performing some analyses about their sales in Europe.

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

1. *Car models with an increasing number of sales in Italy in the last two years*. The application considers only the sales in Italy and selects the car models with the number of sales in year 2020 greater than the number of sales in year 2019. Store in the output HDFS folder the identifiers (ModelIDs) of the selected car models and the difference between the sales in 2020 and the sales in 2019 for each of the selected car models (the output contains one line for each of the selected car models and the format is ModelID\tNum.sales_2020- Num.sales_2019).

Suppose that the input is SalesEU.txt and it has been already set and also 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** (19 points)

The managers of PoliCars are interested in performing some analyses related to car sales.

The managers of PoliCars asked you to develop one single application to address all the analyses they are interested in. The application has five arguments: the three input files CarModels.txt, SalesEU.txt, SalesExtraEU.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 Python code, to address the following points:

1. *Car models manufactured by FIAT with a high number of sales in Italy and a high average price in Italy.* The first part of this application considers only the car models manufactured by FIAT and the sales related to Italy (the set of sales related to Italy is the subset of the sales stored in SalesEU.txt associated with Country equal to 'Italy'. There are no sales related to Italy in SalesExtraEU.txt). Given only the Italian sales of the car models manufactured by FIAT, the application selects the car models associated with more than 1000000 sales in Italy and an average price of sale greater than 50000 euro in Italy. The application stores in the first HDFS output folder the identifiers (ModelIDs) of the selected car models (one car model per output line).

2. *Car models that were always characterized by an increasing annual sales trend.* Considering all sales (i.e., the union of the sales in SalesEU.txt and SalesExtraEU.txt) the application selects the car models that **were always characterized** by an increasing annual sales trend. A car model is always characterized by an increasing annual trend if for all windows of two consecutive years the number of sales of the considered car model in the first year of the window is less than the number of sales of the same car model in the second year of the window. The application stores in the second HDFS output folder the identifiers (ModelIDs) of the selected car models (one car model per output line).

   Note: Assume that each car model was sold at least one time in each year.

   **Examples related to Part 2 of this exercise.**

   For the sake of simplicity, in the following examples we suppose there are only four years (2010, 2011, 2012, and 2013) but pay attention that the input files contain data related to more than 50 years and your application must return the correct result considering all data.

   Suppose the following table contains the number of annual sales for the car models Model10, Model20, and Model30 (in the input data there are several more car models).

| | 2010 | 2011 | 2012 | 2013 |
|---|---|---|---|---|
| Model10 | 50,000 | 100,000 | 105,000 | 110,000 |
| Model20 | 52,000 | 40,000 | 95,000 | 35,000 |
| Model30 | 45,000 | 45,000 | 50,000 | 60,000 |

Given the example data reported in the above table, only Model10 is selected and stored in the second output folder (Model10 is always characterized by an increasing annual sales trend) whereas Model20 and Model30 are discarded because each of them is characterized by at least two consecutive years with a non increasing sales trend.

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

```
carModelsPath = 'data/CarModels.txt'
salesEUPath = 'data/SalesEU.txt'
salesExtraEUPath = 'data/SalesExtraEU.txt'

output1 = 'out1'
output2 = 'out2'
```