Exercises Big Data Management

Database Technologies and Information Management (DTIM) group
Universitat Politècnica de Catalunya (BarcelonaTech), Barcelona
May 28, 2025

Contents

0.1 Spark in use

0.1.1 Problems

1. Consider a file (*wines.txt*) containing the following data:

   wines.txt
   type_1,2.064254
   type_3,2.925376
   type_2,2.683955
   type_1,2.991452
   type_2,2.861996
   type_1,2.727688

   Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve the minimum value per type. Do not use SQL and minimize the use of other Python libraries or code. Save the results in *output.txt*.

2. Consider two files containing the following data:

   Employees.txt
   EMP1;CARME;400000;MATARO;DPT1
   EMP2;EUGENIA;350000;TOLEDO;DPT2
   EMP3;JOSEP;250000;SITGES;DPT3
   EMP4;RICARDO;250000;MADRID;DPT4
   EMP5;EULALIA;150000;BARCELONA;DPT5
   EMP6;MIQUEL;125000;BADALONA;DPT5
   EMP7;MARIA;175000;MADRID;DPT6
   EMP8;ESTEBAN;150000;MADRID;DPT6

   Departments.txt
   DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
   DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
   DPT3;MARKETING;1;PAU CLARIS;BARCELONA
   DPT4;MARKETING;3;RIOS ROSAS;MADRID
   DPT5;VENDES;1;MUNTANER;BARCELONA
   DPT6;VENDES;1;CASTELLANA;MADRID

   Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve for each employee his/her department information. Do not use SQL and minimize the use of other Python libraries or code. Save the results in *output.txt*.

3. Consider an error log file (*log.txt*) like the one bellow:

   log.txt
   20150323;0833;ERROR;Oracle
   20150323;0835;WARNING;MySQL
   20150323;0839;WARNING;MySQL
   20150323;0900;WARNING;Oracle
   20150323;0905;ERROR;MySQL
   20150323;1013;OK;Oracle
   20150323;1014;OK;MySQL
   20150323;1055;ERROR;Oracle

   Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve the lines corresponding to both *errors* and *warnings*, but adding *Important:* at the beginning of the identifier of those of errors (i.e., only *errors*). Do not use SQL and minimize the use of other Python libraries or code. Save the results in *output.txt*.

4. Assume that "spark" variable is a **Spark** session and the `dataset.csv` contains the two columns "Color" and "Radius". Clearly **identify the problems** you find in the following Spark code and propose some fix to obtain the expected result.

```
df0= spark.read.csv("dataset.csv", \
                    header='true', \
                    inferSchema='true', \
                    sep=',')
df1 = df0.select("Color", "Radius")
df2 = df1.withColumn("Pi", 3.141592)
df3 = df2.withColumn("Area", df2.Radius*df2.Radius*df2.Pi)
df4 = df3.groupBy("Color").max("Area")
df5 = df4.sort("Color")
```

5. Given two files containing the following kinds of data:

Employees.txt with fields: EmployeeID; EmployeeName; YearlySalary; CityOfResidence; SiteOfWork

EMP4;RICARDO;250000;MADRID;DPT4
EMP5;EULALIA;150000;BARCELONA;DPT5
EMP6;MIQUEL;125000;BADALONA;DPT5
EMP7;MARIA;175000;MADRID;DPT6
EMP8;ESTEBAN;150000;MADRID;DPT6
...

Departments.txt with fields: SiteID; DepartmentName; StreetNumber; StreetName; City

DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
DPT3;MARKETING;1;PAU CLARIS;BARCELONA
DPT4;MARKETING;3;RIOS ROSAS;MADRID
...

Consider the following **PySpark code** and answer the questions bellow.

```
source1 = spark.read.format("csv").load("employees.txt", header='false', inferSchema='true', sep=";")
source2 = spark.read.format("csv").load("departments.txt", header='false', inferSchema='true', sep=";")
A = source1.toDF("eID","eName","eSalary","eCity","eDpt")
B = source2.toDF("dID","dArea","dNumber","dStreet","dCity")
C = A.select(A.eCity.alias("city"))
D = B.select("dArea")
E = D.crossJoin(C)
F = B.select("dArea",B.dCity.alias("city"))
G = E.subtract(F)
H = G.select("dArea")
result = D.subtract(H)
```

(a) State in natural language the corresponding query it would answer?

(b) Clearly indicate any mistake or improvement you can fix/make in the code? For each of them give (1) the line number, (2) pseudo-code to implement the fix, and (3) brief rationale.

6. Given two files containing the following kinds of data:

Employees.txt with fields: EmployeeID; EmployeeName; YearlySalary; CityOfResidence; SiteOfWork

EMP1;RICARDO;250000;MADRID;DPT1
EMP2;EULALIA;150000;BARCELONA;DPT2
EMP3;MIQUEL;125000;BADALONA;DPT3
EMP4;MARIA;175000;MADRID;DPT4
EMP5;ESTEBAN;150000;MADRID;DPT3
...

Departments.txt with fields: SiteID; DepartmentName; StreetNumber; StreetName; City

DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
DPT3;MARKETING;1;PAU CLARIS;BARCELONA
DPT4;MARKETING;3;RIOS ROSAS;MADRID
...

Give a sequence of Spark operations in pseudo-code (resembling PySpark) to obtain for each city where employees that work in a site of a department in Barcelona live, the sum of the salaries of those employees. The result for the exemplary data would be:

MADRID;400000

BADALONA;125000

...

7. Consider three files containing the following kinds of data:

Employees.txt
EMP1,CARME,400000,MATARO,DEPT1,PROJ1
EMP2,EULALIA,150000,BARCELONA,DEPT2,PROJ1
EMP3,MIQUEL,125000,BADALONA,DEPT1,PROJ3

Projects.txt
PROJ1,IBDTEL,TV,1000000
PROJ2,IBDVID,VIDEO,500000
PROJ3,IBDTEF,TELEPHONE,200000
PROJ4,IBDCOM,COMMUNICATIONS,2000000

Departments.txt
DEPT1,MANAGEMENT,10,PAU CLARIS,BARCELONA
DEPT2,MANAGEMENT,8,RIOS ROSAS,MADRID
DEPT4,MARKETING,3,RIOS ROSAS,MADRID

Provide the ordered list of Spark operations (no need to follow the exact syntax, but just the kind of operation and main parameters) you would need to obtain the departments with all employees assigned to the same project. The result must include department number. Save the results in `output.txt`. In the previous example, the result should be *DEPT2* and *DEPT4*.

8. Consider two files containing the following kinds of data:

Employees.txt
EMP4;RICARDO;250000;MADRID;DPT4
EMP5;EULALIA;150000;BARCELONA;DPT5
EMP6;MIQUEL;125000;BADALONA;DPT5
EMP7;MARIA;175000;MADRID;DPT6
EMP8;ESTEBAN;150000;MADRID;DPT6

Departments.txt
DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
DPT3;MARKETING;1;PAU CLARIS;BARCELONA
DPT4;MARKETING;3;RIOS ROSAS;MADRID

Provide the ordered list of Spark operations (no need to follow the exact syntax, but just the kind of operation and main parameters) you'd need to retrieve the list of department IDs for those departments with workers from all cities where there are employees. Save the results in `output.txt`.

9. Consider two files containing the following kinds of data:

Employees.txt
EMP1;RICARDO;250000€;MADRID;SITE2
EMP2;EULALIA;150000€;BARCELONA;SITE1
EMP3;MIQUEL;125000€;BADALONA;SITE3
EMP4;MARIA;175000€;MADRID;SITE2
EMP5;ESTEBAN;150000€;MADRID;SITE4

Departments.txt
SITE1;DPT.MANAGEMENT;FLOOR10;ST.PAU CLARIS;BARCELONA
SITE2;DPT.MANAGEMENT;FLOOR8;ST.RIOS ROSAS;MADRID
SITE3;DPT.MARKETING;FLOOR1;ST.PAU CLARIS;BARCELONA
SITE4;DPT.MARKETING;FLOOR1;ST.RIOS ROSAS;MADRID
SITE5;DPT.MARKETING;FLOOR5;ST.MARTI PUJOL;BADALONA

Provide the ordered list of Spark operations (no need to follow the exact syntax, but just the kind of operation and main parameters) you'd need to retrieve the list of department IDs for those departments with sites in all cities where employees live (these employees can be even from other departments). Save the results in `output.txt`. In the previous example, the result should be *DPT.MANAGEMENT*, because it has sites in all the three cities where there are employees (i.e., MADRID, BARCELONA and BADALONA). However, *DPT.MANAGEMENT* should not be in the result, because it does not have any site in BADALONA, where EMP3 lives.

10. Consider three files relating to a bibliographic database: `author.csv` relates authors with papers (you may assume that author names are unique, that authors have one or more papers, and that papers have one or more authors); `title.csv` gives the title of a paper (you may assume a paper has one title, but one title may be shared by many papers); and `citation.csv` indicates which papers cite which other papers (you may assume that each paper cites at least one other paper, that a paper may be cited zero or more times, and that a paper cannot cite itself).

`author.csv`

AUTHOR	PAPERID
...	...
C. Gutierrez	GP2014
C. Gutierrez	AGP2013
C. Gutierrez	GZ2011
...	...
J. Perez	GP2014
J. Perez	AGP2013
J. Perez	P2017
...	...
R. Angles	AGP2013
R. Angles	AKK2016
...	...

`title.csv`

PAPERID	TITLE
...	...
GP2014	Semantics of SPARQL
AGP2013	Deduction for RDF
GZ2011	Graph databases
...	...

`citation.csv`

CITES	CITED
...	...
GP2014	AGP2013
AGP2013	GZ2011
P2017	AKK2016
...	...

The headers are shown for illustration here. They do not need to be considered.

The count of self-citations for an author A, denoted $\mathsf{self}(A)$, is defined as the number of citation pairs (P_1, P_2) where A is an author of both. The count of citations given by an author A, denoted $\mathsf{give}(A)$, is the count of citation pairs (P_1, P_2) such that A is an author of P_1. The count of citations received by A, denoted $\mathsf{receive}(A)$, is the count of citation pairs (P_1, P_2) where A is an author of P_2. The ratio of self-citations to all citations given and received are then defined, respectively, as $\frac{\mathsf{self}(A)}{\mathsf{give}(A)}$ and $\frac{\mathsf{self}(A)}{\mathsf{receive}(A)}$. In case that $\mathsf{receive}(A) = 0$, you should omit the author A from the results (note that $\mathsf{give}(A)$ cannot be 0 as an author must have at least one paper and a paper must have at least one citation). We provide an example output for the input data:

AUTHOR	SELFGIVERATIO	SELFRECEIVERATIO
C. Gutierrez	1.000	1.000
J. Perez	0.333	1.000
R. Angles	0.000	0.000
...	...	

Headers are only shown for illustration. We will use Apache Spark to perform the analysis and compute the output. You should not assume any ordering of the input files. You do not need to order the output file in any particular way.

Given this input and desired output, design a Spark process to complete the required processing. In particular, you should draw the high-level DAG of operations that the Spark process will perform, detailing the sequence of transformations and actions. You should briefly describe what each step does, clearly indicating which steps are transformations and which are actions. You should also indicate which RDDs are virtual and which will be materialized. You should use caching if appropriate. You should provide details on any functions passed as arguments to the transformations/actions you use.

0.2 Spark internals

0.2.1 Theoretical questions

1. What indicates to Spark query optimizer the end of a stage and the beginning of the next one?

0.2.2 Problems

1. Considering the file and result example, briefly indicate the problems you find in the Spark code below (if any), and **modify the code** to fix them (if needed).

`Employees.txt`
EMP1;CARME;40000;MATARO;DPT1;PROJ1
EMP2;EUGENIA;35000;TOLEDO;DPT2;PROJ1
EMP3;JOSEP;25000;SITGES;DPT3;PROJ2
EMP4;RICARDO;25000;MADRID;DPT4;PROJ2
EMP5;EULALIA;15000;BARCELONA;DPT5;PROJ2
EMP6;MIQUEL;12500;BADALONA;DPT5;PROJ3
EMP7;MARIA;17500;MADRID;DPT6;PROJ3
EMP8;ESTEBAN;15000;MADRID;DPT6;PROJ3

`Expected result`
[PROJ1,[EUGENIA, 35000, 37500.0]]
[PROJ2,[EULALIA, 15000, 21666.6]]
[PROJ3,[MIQUEL, 12500, 15000.0]]

```
rawEmps = sc.textFile(r"Employees.txt")\\
emps = rawEmps.map(lambda l: tuple(l.split(";"))).cache()\\
RDD1 = emps.map(lambda t: (t[5], (int(t[2]), 1)))\\
RDD2 = RDD1.reduceByKey(lambda t1, t2: (t1[0] + t2[0])/(t1[1] + t2[1]))\\
RDD3 = emps.map(lambda t: (t[5], t))\\
RDD4 = RDD3.join(RDD2)\\
RDD5 = RDD4.filter(lambda t: int(t[1][0][2])<t[1][1])\\
RDD6 = RDD5.map(lambda t: (t[0],(t[1][0][1],t[1][0][2],t[1][1])))\\
Result = RDD6.sortByKey()\\
```

2. Consider the following pipeline. This pipeline runs in a 4-machine cluster with HDFS to store the files and Spark to execute it. `File1` and `File2` are distributed in the cluster in 6 chunks each.

```
RddF1:= sc.textFile("...file1.txt")
RddF2:= sc.textFile("...file2.txt")
Rdd2:= RddF1.mapToPair(s.split(";")[0], s.split(";")[1-2])
Rdd3:= Rdd2.GroupByKey()
Rdd4:= Rdd3.MapValues(f1)
Rdd5:= RddF2.mapToPair(s.split(";")[0], s.split(";")[1-2])
Rdd6:= Rdd5.GroupByKey()
Rdd7:= Rdd6.MapValues(f2)
Rdd8:= Rdd4.join(Rdd7)
Rdd8.save("...file3.txt")
```

   a) How many stages will the scheduler generate for this pipeline? (Justify your answer)

b) How many tasks will be generated within each stage? (Justify your answer)

3. Consider the legacy code written in MapReduce that specifies a `map()`, `combine()` and `reduce()` operations. This job reads from a text file f_1. The combine and reduce operations coincide and you can assume all functions are correct. Write a Spark pipeline **equivalent to the MapReduce** job. Use `fmap` and `freduce` to refer to the code executed inside the map and combine / reduce operations. You can parametrise the `fmap` and `freduce` functions but resulting in minimal code adaptation.

4. Given a file of $3.2GB$ stored in an HDFS cluster of 50 machines, and containing $16 \cdot 10^5$ key-value pairs in a SequenceFile; estimate the execution time of a Spark job containing a single map transformation and an action storing the results in a file. Explicit any assumption you make and consider also the following parameters:

   - Chunk size: $128MB$ (default)
   - Replication factor: 3 (default)
   - Map function (i.e., the parameter of the transformation) execution time: 10^{-3}sec/call (this is **the only cost** you have to consider)
   - Save action execution time: 0sec (do not consider its cost at all)