Pràctica 1 - Spark

Maig 2018

Índex

L'objectiu d'aquesta pràctica (tutorial) és fer una introducció al sistema Spark i donar a conèixer alguna de les funcions que hi ha disponibles en aquest entorn per manipular dades. Ens centrarem en fitxers CSV (Comma Separated Values) i realitzarem algunes operacions bàsiques sobre elles. A la següent pràctica realitzarem operacions més avançades.

Al directori d'aquest tutorial disposeu del fitxer `2007.csv`. Aquest fitxer s'ha obtingut de l'adreça web `http://stat-computing.org/dataexpo/2009/the-data.html` i conté informació dels vols en avió que s'han realitzat durant diversos anys. Començarem amb el fitxer `2007.csv`, que conté informació dels vols de l'any `2007.csv` (podeu agafar qualsevol altre fitxer si així ho desitgeu).

En aquest tutorial es proposen exercicis que no fa falta entregar. *Aneu amb molt de compte de copiar i enganxar el codi que hi ha aqui, ja que pot donar a errors d'execució.*

1 Engegar Spark

Per començar hem d'engegar Spark. Per fer-ho obrir un terminal y executar

```
$ pyspark
```

S'engegarà el sistema Spark i podem començar a introduir les instruccions Spark. Spark farà servir la versió de Python que s'engega en executar `python` al nostre terminal. Si volem fer servir un altre intèrpret de Python (podem tenir, per exemple, dues versions de Python intal·lades) cal establir la variable d'entorn `PYSPARK_PYTHON` indicant-t'hi l'executable de Python a fer servir. Per exemple,

```
$ export PYSPARK_PYTHON=/usr/bin/python3
```

Per assegurar que tot està bé, assegurem que hi ha el driver de Spark està engegat. Per això executem

```
>>> spark
```

En executar aquesta instrucció Spark ens ha de retornar la referència a l'objecte del driver.

2 Lectura de fitxers

Spark ens ofereix múltiples funcions per llegir fitxers de diferents tipus: CSV, JSON, ORC, SQL, o fitxers de text. Aquí ens utilitzarem fitxers CSV (Comma Separated Values), que és el format utilitzat pels fitxers que volem utilitzar.

Hem de començar per demanar a Spark que volem operar sobre aquest fitxer. Per fer-ho especifiquem la següent instrucció

```
>>> df = spark.read.format("csv") \
        .option("header", "true") \
        .option("nullValue","NA") \
        .option("inferSchema", "true").load("2007.csv")
```

Les opcions associades a la lectura del fitxer són intuïtives, oi? Aquesta operació demana la lectura del fitxer `2007.csv` i l'assigna al DataFrame `df`. L'opció de `nullValue` permet especificar què és el que hi ha a una columna en cas que la dada no estigui disponible o no es coneix. Per defecte Spark suposa que la columna està buida, però per aquesta base de dades algunes de les columnes per les quals no es coneix el seu valor s'omple amb NA (Not Available).

En el moment d'executar aquesta instrucció no es llegeix realment el fitxer. Sí que s'observarà que Spark fa "una passada" pel fitxer per tal d'analitzar-lo. Ara podem demanar a Spark que ens indiqui el *schema* associat a la taula que hem carregat: en altre paraules, que ens indiqui les columnes que hi ha a la nostra "fulla de càlcul" així com el tipus associat

```
>>> df.printSchema()
```

També li podem demanar a l'Spark el nombre de particions que hi ha associat al fitxer. Sí, tot i que només hi ha un fitxer d'entrada i estiguem executant el mode local l'Spark ens ha "dividit" el fitxer de forma automàtica en particions. Tingueu en compte que si estem executant Spark en un clúster el fitxer d'entrada ja estarà dividit (prèviament) en particions.

Cuantes particions tenim?

```
>>> df.rdd.getNumPartitions()
```

El fet que el fitxer hagi estat dividit en particions permet aprofitar (en mode local) la capacitat multiprocessador de l'ordinador. En realitzar una operació, cada processador pot encarregar-se de realitzar una operació sobre una partició diferent.

3 Operacions bàsiques

Comentem aquí alguna de les operacions bàsiques que es poden realitzar sobre el nostre DataFrame.

3.1 Eliminar i/o columnes

Eliminar columnes d'un DataFrame és senzill i és pot fer amb la transformació `drop`. Es tracta d'una transformació. L'operació encara no es realitza encara realment...

```
>>> df2 = df.drop("FlightNum","TailNum","UniqueCarrier")
```

Podem fer l'operació a la inversa. En comptes d'eliminar columnes, anem a crear un nou DataFrame seleccionant únicament les columnes sobre les quals ens interessa operar. Això ho fem mitjançant la transformació `select`. Mitjançant l'acció `show` podem visualitzar el resultat

```
>>> df2 = df.select("Origin", "Dest", "ArrDelay", "DepDelay")
>>> df2.show()
```

A partir d'ara treballarem amb `df2`. Però abans de fer-ho, hem de "netejar" el DataFrame eliminant totes aquelles files en què hi hagi un NA a la columna `ArrDelay` o `DepDelay`. Ho podem fer així...

```
>>> df3 = df2.na.drop()
```

3.2 Afegir columnes

Podem afegir noves columnes i afegir-hi nova informació. Observar, per exemple, que el DataFrame ens ofereix informació sobre el retard de sortida (`DepDelay`) així com el retard d'arribada (`ArrDelay`). Podem ara afegir una nova columna que sigui la suma dels dos retards.

```
>>> from pyspark.sql.functions import expr
>>> df4 = df3.withColumn("SumDelay", expr("ArrDelay + DepDelay"))
```

De moment només hem realitzat transformacions. Si voleu podem comprovar, mitjançant una acció, que l'operació s'ha fet correctament.

```
>>> df4.select("DepDelay", "ArrDelay", "SumDelay").show(10)
```

Observar que l'acció s'ha fet molt ràpid! Això és perquè només s'ha realitzat l'operació (de moment) sobre els 10 primers registres del nostre DataFrame.

Ens pot interessar saber quin ha sigut el retard mínim i el retard màxim. És fàcil de saber-ho

```
>>> from pyspark.sql.functions import max, min
>>> df4.select(max("SumDelay"),min("SumDelay")).show()
```

En aquest cas cal recórrer tots els elements del DataFrame per saber el valor mínim i màxim del retard total. Quin ha sigut el retard mig?

```
>>> from pyspark.sql.functions import avg
>>> df4.select(max("SumDelay"),min("SumDelay"),avg("SumDelay")).show()
```

És important mencionar que estem operant sobre DataFrames. Per poder assignar els valors mínim, màxim i valors mitjos a variables Python cal transformar el DataFrame a un tipus de Python. Ho veurem més endavant, a la secció 3.7.

3.3 Emmagatzematge temporal de dades

A partir d'ara farem servir sovint el DataFrame `df4` per realitzar operacions sobre ell. Spark, cada cop que realizem una operació sobre aquest DataFrame, construirà el graf de transformacions a realitzar des del principi, des de la primera instrucció de lectura que hem introduït.

Per qüestions d'eficiència, això no és necessari. Podem indicar-li l'Spark que emmagatzemi temporalment (sigui a memòria o a disc) el DataFrame `df4`. D'aquesta forma, totes les subsegüents operacions que es realitzin sobre `df4` seran més ràpides, ja que Spark farà les operacions a partir de `df4` i no `df`, el primer DataFrame.

```
>>> df4.cache()
```

3.4 Operacions de filtratge

L'operació de filtratge ens permet crear un nou DataFrame que compleixi una certa condició. Hi ha dues funcions que permeten fer-ho: `where` i `filter`. Començarem per fer servir la funció `where` atesa la seva similitud (tal com diu la bibliografia) amb la funció SQL corresponent.

```
>>> df5 = df4.where("SumDelay < 0")
>>> df5.show()
```

S'ha dividit l'operació en dos passos: la primera és la transformació, i la segona és l'acció. L'acció `show` només ens mostra els primers elements de la llista, però quants elements n'hi ha? Això ho podem fer amb l'acció `count`. Comparem-ho amb el nombre de registres originals que tenim.

```
>>> df3.count()
>>> df5.count()
```

Hi ha molts vols que arriben abans d'hora a destinació!

Podem aplicar múltiples filtratges a un DataFrame. Per exemple, podem preguntar-nos si els avions que surten de Nova York arriben a destinació abans d'hora (observeu les comentes simples a JFK).

```
>>> df5 = df4.where("SumDelay < 0").where("Origin == 'JFK'")
>>> df5.show()
>>> df5.count()
```

L'altre funció de filtratge disponible es diu `filter`. És molt similar a la funció `where`, amb l'avantatge que es poden fer servir varibles de Python per establir els criteris sobre les condicions de filtratge. Anem a implementar el mateix filtre de fa un moment però ara amb la transformació `filter`.

```
>>> from pyspark.sql.functions import col
>>> i = 0
>>> city = "JFK"
>>> df5 = df4.filter(col("SumDelay") < i).filter(col("Origin") == city)
>>> df5.count()
```

Exercici

Atesos els exemples que hem vist, ens podem preguntar: donat l'aeroport d'origen de Nova York, quin és el retard total (columna `SumDelay`) mínim, màxim i mig associat? Podeu provar també la funció `sum`, una transformació que realitzarà la suma de la columna que especifiqueu.

```
>>> from pyspark.sql.functions import sum
>>> <omplir amb el vostre codi>
```

3.5 Operacions d'ordenació

Podem ordenar les dades pel retard total

```
>>> from pyspark.sql.functions import asc, desc
>>> df5 = df4.sort(asc("SumDelay"))
```

En aquest cas es realitza una operació d'ordenació ascendent. De nou, observar que sort és una transformació. No es realitza l'operació d'ordenació en el moment d'introduir-la. Hem de demanar que, per exemple, volem visualitzar les dades.

```
>>> df5.show()
```

Podem fer l'ordenació de forma `descendent`. En aquest cas, a més, el DataFrame que es crea no tindrà tots els elements de la taula, sinó que el DataFrame només contindrà els primers 5 elements.

```
>>> df5 = df4.sort(desc("SumDelay")).limit(5)
>>> df5.show()
```

L'operació d'ordenació, de forma similar a les operacions de filtratge que s'han realitzat anteriorment, s'han fet realitzant múltiples processadors ja que el DataFrame d'entrada té múltiples particions.

3.6 Obtenir els elements únics

Us heu preguntat quants aeroports diferents d'origen hi ha? Això és fàcil de saber mitjançant la transformació `distinct`.

```
>>> df5 = df4.select("Origin").distinct()
>>> df5.count()
```

Exercici

Quantes destinacions diferents hi ha? A més, què és el que fa la següent operació?

```
>>> df4.select("Origin","Dest").distinct().count()
```

3.7 Accés a les dades des de Python

Hem vist com podem fer operacions amb els DataFrames fent servir l'entorn Spark. Amb Spark no es pot accedir de forma individual als registres del DataFrame per manipular-los.

En determinats casos, però, ens pot interessar accedir als resultats de les operacions realitzades fent servir les estructures de dades que ens ofereix Python. D'aquesta forma podrem manipular de forma individual els registres resultants. Això es pot fer mitjançant l'acció `collect`, la qual recull totes les particions del DataFrame i les transfereix al driver. Cal anar amb compte en fer aquesta operació en un clúster d'ordinadors, ja que cal assegurar-se que les dades caben a la memòria del driver.

Anem a veure-ho: agafem el DataFrame `df4` (que conté milers d'elements), limitem el nombre de dades a recollir (per evitar problemes) i les recollim al driver amb l'acció `collect`. Un cop recollides, podem accedir als elements fent servir instruccions de Python.

```
>>> dades = df4.limit(5).collect()
>>> dades
>>> dades[0]
>>> dades[0][3]
```

4 Escriptura de fitxers

Hem vist com llegir dades d'un fitxer CSV de disc. Aquí descriurem el procés d'escriptura. En particular, ens centrarem només en fitxers CSV.

Suposem que volem escriure el DataFrame `df4` a disc. Recordem el nombre de particions que té

```
>>> df4.rdd.getNumPartitions()
```

Per defecte, si ara escrivíssim les dades a disc, es crearien tants fitxers com particions hi ha. Estem treballant en mode local i en aquest cas ens interessa crear un únic fitxer. Per això cal re-particionar el DataFrame perquè estigui compost només per una sola partició.

```
>>> df4_one = df4.coalesce(1)
>>> df4_one.rdd.getNumPartitions()
```

El DataFrame `df4_one` té només una partició!

Com emmagatzemar un DataFrame a disc? És senzill...

```
>>> df4_one = spark.write.format("csv") \
        .option("header", "true") \
        .save("df4_one.csv")
```

Ara podeu sortir de l'aplicació i comprovar que, efectivament, les dades han sigut desades a disc.

Exercici

És interessant que feu un experiment: abans hem ordenat els elements de `df4` pel valor de `SumDelay`. Intenteu ordenar ara els valors de `df4_one`. Hi haurà alguna diferència en l'eficiència de l'ordenació? Feu servir alguna eina gràfica que us mostri el consum de CPU per veure-ho. Per a què és útil tenir les dades dividides en particions?