TP Plans et partitions BDLE 2021

Version téléchargée par les étudiants

date de révision du document :

Préparation

installation spark

Classe utilitaire : Mesure

Sert pour mesurer la quantité de shuffle lors de l'exécution d'un requête

L'objet mesure *m* servira à mesurer les requêtes

Fonctions pour afficher le contenu des partitions d'un dataframe

On définit

- showPartitions : affiche les *n* premiers éléments de chaque partition
- showPartitionSize : affiche le nombre d'éléments dans chaque partition

```
# fonction auxilliaire
def partSize(partID, iterateur):
  c=0
  suivant = next(iterateur, None)
  while suivant is not None :
    c+=1
    suivant = next(iterateur, None)
  return [(partID, c)]


def showPartitionSize(df):
  t = df.selectExpr("1").rdd.mapPartitionsWithIndex(partSize)
                            llect():
                          , ":", nbElt, "éléments")
  print()

```

Le notebook a bien été enregistré.

```
    while suivant is not None and c < N :
      c+=1
      head.append(suivant)
      suivant = next(iterateur, None)
    return [(partID, head)]
  t = df.rdd.mapPartitionsWithIndex(topN)
  for (partID, head) in t.collect():
    print("Partition", partID, ",", size[partID], "éléments")
    for row in head:
        print(row)
    print()

print('showPartitions et showPartitionSize définies')
```

```
showPartitions et showPartitionSize définies
```

Définir les dataframes associés aux données

Les **films**

```
films = spark.read.json(dir + "films.json").selectExpr("nF", "titre", "g as gen
display(films)
```

Les **notes**

```
notes = spark.read.json(dir + "notes1M.json").selectExpr("nF", "nU", "note", "a
display(notes)
```

Exercice 1 : partitionnement des données en MEMOIRE

Une des particularités de Spark est la possibilité de charger et de “maintenir” (persister) les données en **mémoire** vive. Cette fonctionnalité permet d'accélérer les accès fréquents aux données en évitant de relire plusieurs fois des données stockées sur disque.

Nettoyer l'espace mémoire (vider le cache) en début d'exercice

Le notebook a bien été enregistré.

- FileScan
- Exchange hashpartitioning(annee, *N*)

Rmq: **repartition** est une transformation et non une action. Le (re)partitionnement est défini mais n'est pas encore exécuté

```
1 notes.rdd.getNumPartitions()
```

```
2
```

```
1 notes.explain()
```

```
== Physical Plan ==
*(1) Project [nF#29L, nU#30L, note#31, annee#27L]
+- FileScan json [annee#27L,nF#29L,nU#30L,note#31] Batched: false, DataFilter
```

```
1 N=4
2 notesMemoire_par_annne = notes.repartition(N, "annee")
3 notesMemoire_par_annne.explain()
```

```
== Physical Plan ==
Exchange hashpartitioning(annee#27L, 4), REPARTITION_BY_NUM, [id=#50]
+- *(1) Project [nF#29L, nU#30L, note#31, annee#27L]
   +- FileScan json [annee#27L,nF#29L,nU#30L,note#31] Batched: false, DataFil
```

Rendre ce dataframe persistant en mémoire. Rmq: ici l'action **count** exécute le repartionnement défini ci-dessus.

```
1 notesMemoire_par_annne.persist()
2 notesMemoire_par_annne.count()
```

```
1301573
```

Le notebook a bien été enregistré.

Contenu initial des partitions du dataframe Notes

```
1 showPartitions(notesMemoire_par_annne)
```

Accès aux données cachées

Comprendre le plan d'accès :

- InMemoryRelation : cache mémoire
- InMemoryTableScan : accès au cache

Rmq: les autres opérateur en dessous de InMemoryRelation ne sont précisés qu'à titre d'info rappelant l'historique de création du cache, ils ne sont pas exécutés

```
1 notesMemoire_par_annne.explain()
```

```
== Physical Plan ==
Exchange hashpartitioning(annee#27L, 4), REPARTITION_BY_NUM, [id=#50]
+- *(1) Project [nF#29L, nU#30L, note#31, annee#27L]
   +- FileScan json [annee#27L,nF#29L,nU#30L,note#31] Batched: false, DataFil
```

Durée de **lecture** des données cachées

```
1 # A compléter
```

Le notebook a bien été enregistré.

```
 6 r1 = notes.groupBy("annee").agg(count("note").alias("nb"))#, avg("note").alias(
 7 r1.explain()
 8
 9 # notes.createOrReplaceTempView("NOTES")
10 # r1 = spark.sql("""
11 # select annee, count(*) as nb, avg(note) as moyenne
12 # from NOTES
13 # group by annee
14 # """)
```

```
== Physical Plan ==
*(2) HashAggregate(keys=[annee#27L], functions=[count(note#31)])
+- Exchange hashpartitioning(annee#27L, 4), ENSURE_REQUIREMENTS, [id=#854]
   +- *(1) HashAggregate(keys=[annee#27L], functions=[partial_count(note#31)]
      +- *(1) ColumnarToRow
         +- InMemoryTableScan [note#31, annee#27L]
               +- InMemoryRelation [nF#29L, nU#30L, note#31, annee#27L], Stor
                     +- *(1) Project [nF#29L, nU#30L, note#31, annee#27L]
                        +- FileScan json [annee#27L,nF#29L,nU#30L,note#31] Ba
```

```
1 display(r1)
```

Que font ces opérations ?

Le notebook a bien été enregistré.

```
            +- InMemoryTableScan [nU#30L, note#31]
                  +- InMemoryRelation [nF#29L, nU#30L, note#31, annee#27L], Stor
                        +- *(1) Project [nF#29L, nU#30L, note#31, annee#27L]
                           +- FileScan json [annee#27L,nF#29L,nU#30L,note#31] Ba
```

Question 5 : Requête GROUP BY sur des données déjà partitionnées

```
1 r2 = notesMemoire_par_annne.groupBy("annee").agg(count("note"))
2 r2.explain()
```

```
   == Physical Plan ==
   *(1) HashAggregate(keys=[annee#27L], functions=[count(note#31)])
   +- *(1) HashAggregate(keys=[annee#27L], functions=[partial_count(note#31)])
      +- *(1) ColumnarToRow
         +- InMemoryTableScan [note#31, annee#27L]
               +- InMemoryRelation [nF#29L, nU#30L, note#31, annee#27L], Storage
                     +- Exchange hashpartitioning(annee#27L, 4), REPARTITION_BY_
```

Le notebook a bien été enregistré.

```
    res={}
    for row in iterateur :
        if not res.get(row.annee):
          res[row.annee]=(row.cums[0],row.cums[1])
        else:
          sum_notes,nb_notes=res[row.annee]
          res[row.annee] = (sum_notes+row.cums[0],nb_notes+row.cums[1])
    return [(annee,cums[1],cums[0]/cums[1]) for annee,cums in res.items()]

return  df.rdd.mapPartitionsWithIndex(mean_map).partitionBy(4, lambda x : x).
```

```
avg_notes=Myagg(notes)
```

Le notebook a bien été enregistré.

pour faire le tri sur chaque partition puis il y aura pas d'autres traitements sur le resultat.

```
1 # COMPLETER
2 notesMemoire_par_annne.orderBy("annee").explain()
```

```
== Physical Plan ==
```

Le notebook a bien été enregistré.

Le notebook a bien été enregistré.

Le notebook a bien été enregistré.

Le notebook a bien été enregistré.

Le notebook a bien été enregistré.

Le notebook a bien été enregistré.

Le notebook a bien été enregistré.

Le notebook a bien été enregistré.