UPMC 2014-2015
UFR 919 M2 BDLE

TME 2 : Jointures paralléles sous Spark

Prérequis :
Consulter la rubrique Informations pratiques/Notes sur les TD-TME de la page de l'UE pour les instructions de démarrage.
Pour ce TME, se connecter au serveur Spark distant (instructions de ls sous-rubrique Utilisation à distance).
Les identifiants et mots de passe sont communiqués pendant le TME.
Prendre la peine de vérifier l'interface web après exécution de chaque instruction.
Noter les observations.

Note :
Une rubrique Questions Fréquentes est en cours de construction.
Vous pouvez la consulter et participer à l'enrichir en me suggérant des questions.

Illustration de quelques concepts :

1- FlatMap

def flatMap[B](f: (A) ⇒ GenTraversableOnce[B]): Seq[B]

Applatit une séquence en appliquant une fonction aux éléments de la séquence.
Exemple :

```
val x = Map("a" -> List(11,111), "b" -> List(22,222))

scala> x.flatMap(x=>x._1)
res1: scala.collection.immutable.Iterable[Char] = List(a, b)
```

on peut également ecrire x.flatMap(_._1)

```
scala> x.flatMap(_._2)
res2: scala.collection.immutable.Iterable[Int] = List(11, 111, 22, 222)
```

Expliquer le résultat de

```
val ys = Map("a" -> List(1 -> 11,1 -> 111), "b" -> List(2 -> 222,2 -> 22)).flatMap(_._1)
```

et

```
val ys = Map("a" -> List(1 -> 11,1 -> 111), "b" -> List(2 -> 222,2 -> 22)).flatMap(_._2)
```

Pour vous aider, un indice se trouve dans cette instruction

```
val ys = Map("a" -> List(1 -> 11), "b" -> List(2 -> 22,2 -> 222,3->333, 2->0)).flatMap(_._2)
```

FlatMap et les chaînes de caractères

soit la variable pers (pour personnes)

```
var pers = Array( (1,"pierre"), (5,"alice"), (4, "paul"))
```

Expliquer le résultat de

```
var v=pers.flatMap(x=>x._2)
```

Commentez l'instruction

```
var v=pers.flatMap(x=>x._1)
```

2- Composition de Map

soit la variable pers déclarée ci-dessus

Expliquer le comportement des instructions suivantes

```
pers.flatMap(x=>(x._2) map(y=>y))
pers.flatMap(x=>(x._2) map(y=>x._1))
pers.flatMap(x=>(x._2) map(y=>x._2))
```

ainsi que quelques unes non-valides

```
pers.flatMap(x=>(x._1) map(y=>y))
pers.flatMap(x=>(x._2) map(y=>y._1)))
```

3- Jointures (sur données de taille réduite)

En supposant toujours la même liste de personnes pers, nous définissons une liste visite comme suit

```
var visite = Array((5,"Paris"), (1,"Paris"), (1,"Marseille"), (5,"Lyon"))
```

Nous voulons effectuer une jointure entre ce qu'on peut considérer comme la table des personnes (renseignées par pers) et celle des visites effectuées par ces personnes (renseignées par visite). L'attribut en commun est le pemier de chaque variable, i.e l'entier qui sert d'identifiant.

- Proposer une solution en utilisant les fonctions map, flatMap et filter.
- Proposer une solution à base de RDD en utilisant les fonctions map, filter, union et groupByKey.

Exerice :

Cet exercice reprend l'exercice 2 de la planche précédente dans le but d'illustrer les traitements parallèles sous Spark.

Dans un premier temps, charger le fichier /user/etudiant/ratings.dat en fixant le nombre de partitions à 4.
Rappel de la signature de la fonction textFile de SparkContext.

```
def textFile(path: String, minPartitions: Int): rdd.RDD[String]
```

Effectuer une action vous permettant de charger les données et observer la lecture parellèle des 4 paritions sur l'interface web.

Saisissez la ligne ci-dessous si vous voulez que vos données résident en mémoire.
Attention : possibilité de saturer la mémoire ce qui peut faire planter le spark-shell.

```
var storageLevel = org.apache.spark.storage.StorageLevel.MEMORY_AND_DISK_SER
```

puis appeler la fonction persist() sur la RDD f avec storageLevel comme argument.
Effectuer une action vous permettant de charger les données et observer la création des partitions sur l'onglet Storage de l'interface graphique.

La première question a pour objectif de préparer les données en y appliquant le même traitement que celui de l'exercice 2 de la planche précédente à ceci près qu'**il faudra appliquer la fonction persist() à chaque étape puis les actions habituelles permettant de voir le résultat sur l'interface graphique (onglets Stages et Storage).**

Question 1 - Transformations des données et statistiques.
a- Produire à partir de f une RDD c1 obtenue en transformant chaque ligne en un tableau de chaines.
b- Transformer c1 en c2 de sorte à ce que chaque élément du tableau c1 devienne un n-uplet dans c2.
c- Nous désirons compter le nombre de films par utilisateur.
Rappel : chaque ligne de ratings.dat est de la forme UserID::MovieID::Rating::Timestamp.
En supposant qu'un utilisateur (UserID) note un film une fois au plus, écrire l'opération qui permet d'effectuer l'opération de comptage.
Remarque : l'opération que vous proposerez doit être paramétrée pour permettre un certain degré de parallélisme. Ce dernier est appelé *numTasks* dans la documentation accessible depuis

http://spark.apache.org/docs/latest/programming-guide.html

Qu'observez-vous de nouveau sur l'interface graphique par rapport aux précédentes questions? Comment l'expliquez-vous?

Quel est le degré de parallélisme utilisé par défaut?
Comment retrouver l'utlisateur ayant noté le plus grand nombre de films?

Question 2 - Jointures

Pour cette question, nous allons avoir besoin de charger le fichier /user/etudiant/users.dat.
Rappel : chaque ligne de users.dat est de la forme UserID::Gender::Age::Occupation::Zip-code.
Naturellement, il vous est demandé d'appliquer le même traitement que dans la question 1-a et b dans le but de produire une RDD de nuplets que vous applerez d2 et qui aura le même schéma que c2.
Vous découperez les données en 2 fragments par exemple.

a- Exprimer les opérations permettant de construire les nuplets (film,age) indiquant pour chaque film les âges des utilisateurs qui l'ont notés.

b- Construire et affecter à une variable USER les tuples (utilisateur,age) à partir de d2.
Faire de même pour NOTE cette fois en y stockant les tuples (utilisateur,Rating) à partir de c2 calculée en Question 1.

c- Exprimer la jointure naturelle entre USER et NOTE.