M2 Logicels – 2017-18 – Calcul parallèle
1h45 – Aucun document autorisé

1. Cours (5pts)

1) Citer les principaux avantages de l'exploitation de Spark par rapport à un moteur MapReduce du type Apache Hadoop.
2) Dans l'article 'MapReduce and Parallel DBMSs: Friends or Foes?', quelles sont les qualités/cas d'usage identifiées pour MapReduce?
3) Expliquer la différence entre transformations et actions dans le contexte des opérations sur Spark? Fournir au moins 2 opérations de l'API Spark pour transformation et action
4) A quoi correspond l'étape de shuffle dans le contexte de MapReduce
5) En quoi le caractère lazy des RDD aide à l'optimisation de DAG

2. Spark (8pts)

On considère les données sur des films et les avis émis par des utilisateurs.
Avis (pseudo, film, étoile) // *film* est le numéro du film
Film (num, titre, année, catégorie)

Tous les attributs sont des nombres entiers (type Long) sauf le titre et le pseudo qui sont de type String.
Les données sont stockées dans les collections suivantes :

```
val avis :RDD[(String, Long, Long)]
val film : RDD[(Long, String, Long, Long)]
```

Rappel des opérations sur les collections :
c1.distinct : retourne un ensemble contenant les éléments de c1 sans doublons.
c1.filter(p) : retourne les éléments *e* tels que *e* est dans *c1* et *p(e)* est vrai.
c1.join(c2) retourne les éléments (k, (v, w)) tels que (k, v) est dans *c1* et (k, w) est dans *c2*.
c1.reduceByKey(f) : retourne les éléments (k, y) tels que pour l'ensemble des couples (k, v_i) dans *c1* ayant la clé k, on obtient y en appliquant *f* sur les v_i. On a y = f(...f(f(f(v_1, v_2), v_3), ...), v_n)
Remarque : les opérations join et reduceByKey ne sont applicables que sur des collections contenant des couples. Lorsque nécessaire, transformer préalablement les nuplets en couple à l'aide d'une opération map.

1) Que représente a ?

```
val a = avis.filter({case (pseudo, film, étoile) => étoile == 5}).map({case (pseudo, film, étoile) => film}).distinct
```

2) Que représente b3? Préciser aussi le type des éléments de b3.

```
val b1 = avis.map({case (pseudo, film, étoile) => (film, étoile) })
val b2 = film.map({case (num, titre, année, catégorie) => (num, catégorie) })
val b3 = b1.join(b2).map({case (num, (etoile, catégorie)) => (catégorie,1)}).reduceByKey( (v, w) => v+w)
```

3) Exprimer *d* contenant les pseudos des utilisateurs ayant posté au moins 20 avis. Le type du résultat est RDD[String].

4) Exprimer *u* contenant les utilisateurs (fournir le pseudo) qui ont noté au moins un film qu'Alice a noté.
Remarque: décomposer la réponse en exprimant d'abord *fa* les numéros de films qu'Alice a noté.

5) Les données sont réparties sur 10 machines (dénotées m1 à m10), en utilisant les fonctions de hachage h et h'. Pour $1 \leq i \leq 10$, la machine mi contient les collections Ai et Fi (respectivement des avis et des films) de manière à ce que les fonctions h et h' aient respectivement le pseudo et le numéro du film comme clé de hachage et la valeur I comme résultat.
Soit la requête R suivante : «Quelles personnes ont attribué 5 étoiles à un film de catégorie 1 ?». R est une collection de pseudo, sans doublon. Le résultat de R peut demeurer réparti sur plusieurs machines (inutile de rassembler les données du résultat sur une seule machine).

Quelles données sont transférées entre les machines lors du calcul de R? Expliquer brièvement les étapes. On ne demande **pas** le détail des expressions spark.

6) On modifie la façon dont les avis sont répartis. Les avis sont maintenant répartis par numéro de film :
Ai : les avis tels que h'(film) = i
Les transferts sont-ils les mêmes que dans la question précédente ? Justifier.

3. Spark SQL et Dataframes (4 pts)

On considère le même contexte de données sur des films et avis que dans la question précédente.

1) Dans le programme ci-dessous, expliquer les lignes 1,2,3,4,7,8,9 et 10

```
1. val v1 = sc.textFile("/home/oliv/cours/2015-16/M2/avis")
2. val v2 = v1.map(x=>x.split("\t"))
3. val v3= v2.map(t=>(t(0),t(1).toLong,t(2).toLong))
4. val v4 = v3.toDF("pseudo","film","rate")
5. val f1 = sc.textFile("/home/oliv/cours/2015-16/M2/film").map(x=>x.split("\t")).map(t=>(t(0).toLong,t(1),t(2).toLong,t(3)))
6. val f2 = f1.toDF("num","film","an","cat")
7. val r1 = v4.where(v4("rate")===4 && v4("pseudo")==="bob").select("film","rate")
8. val r2 = f2.where(f2("an")===1982)
9. val r3 = r1.join(r2).select("film","rate")
10. r3.collect
```

2) Traduire la requête de la question précédente en SQL

3) Sans fournir de code, expliquer les instructions qu'il sera nécessaire d'exécuter pour évaluer cette requête SQL avec Spark SQL

4) Sans utiliser d'opérations du DSL de DataFrame autres que celles utilisées dans la question 1 de cet exercice, fournir le numéro des films ayant reçu exactement 100 avis. Remarque: vous pouvez effectuer un pré-traitement sur les RDD avant de réaliser la requête sur le DataFrame.

4. Réflexion (3 pts)

Dans un domaine d'application de votre choix, décrire une fonctionnalité nécessitant l'exploitation des composants de Spark suivants: Spark Core (avec RDD), GraphX, SparkSQL ou DataFrame (dans l'ordre que vous voulez). Pour chaque étape de votre scénario, justifier et décrire (sans écrire de code) l'usage de chacun de ces composants.