Nom :	Prénom :	*page 1*

BDLE – 5I852 - Examen réparti du 13 février 2015

Version CORRIGEE

Ex1 :	
Ex2 :	

Seuls les documents de cours et de TD sont autorisés – Durée : 2h.

Répondre aux questions sur la feuille du sujet dans les cadres appropriés. Utiliser le dos de la feuille précédente si la réponse déborde du cadre. Le barème est donné à titre indicatif. La qualité de la rédaction sera prise en compte. Ecrire à l'encre bleue ou noire. Ne pas dégrafer le sujet. Eteindre et ranger tout téléphone et autre appareil électronique.

Exercice 1. Manipulation de données avec Spark — 6 pts

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, categorie)

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
```

Les numéros des films ayant au moins un avis 5 étoiles

2) Que représente b3? Donner 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)) => (categorie,1)}).reduceByKey( (v, w) => v+w)
```

```
b3
```

b3 contient des couples (catégorie, nbavis) :

Le nombre d'avis pour chaque catégorie

3) Exprimer *d* contenant les numéros d'utilisateurs ayant posté au moins 20 avis. Le type du résultat est RDD[Long].

```
Val d =
```

```
val d = avis.map({case (pseudo, film, étoile) => (pseudo, 1) }).
        reduceByKey( a,b) => a+b).
        filter( {case(pseudo, nbavis) => nbavis >= 20}).
        map({case(pseudo, nbavis) => pseudo})
```

4) Exprimer *u* contenant les utilisateurs qui ont noté au moins un film qu'Alice a noté. Rmq, décomposer la réponse en exprimant d'abord *fa* les numéros de films qu'Alice a noté.

```
Val fa =


Val u =
```

```
val fa = avis.filter({ case(pseudo, film, etoile) => pseudo=='Alice'}).map( {case(p, f, e) => (f, 0)}).distinct

val u = avis.filter({ case(pseudo, film, etoile) => pseudo != 'Alice'}).
            map({ case(pseudo, film, etoile) => (film, pseudo)}).
            join(fa).
            map( { case(f, (pseudo, c)) => pseudo}).distinct
```

5) Les données sont réparties sur les machines M1 à M10 en utilisant les fonctions de hachage h et h'

Mi contient les collections Ai et Fi : ($1 \leq i \leq 10$)

Ai : les avis tels que h(pseudo) = i

Fi : et les films tels que h'(numéro) = i

Soit la requête R suivante : Quelles personnes ont attribué 5 étoiles à un film de catégorie 1 ? R est une collection de pseudo, sans double. R peut être répartie sur les 10 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.

```
Sur Mi on calcule    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Puis on envoie _ _ _ _ _ _ _ _ _ _ _ _ vers _ _ _ _ _ _ _ _

Puis sur Mi on calcule  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Sur Mi on calcule à partir de Fi la liste Li des numéros de films de catégorie 1
Puis on envoie Li vers toutes les autres machines Mj (j différent de i)

Sur Mi on fusionne les Li pour former la liste L
On sélectionne dans Ai les avis 5 étoiles, on obtient Ai5
On fait la jointure de Ai5 avec L sur le numéro de film, et on garde seulement le pseudo

Autre réponse possible

Sur Mi : Commencer par calculer la liste L5 des films ayant un avis 5 étoiles. Puis découper cette liste en fonction de h' : L5i contient les num de film tq h'(num) = i.
Transmettre chaque L5i à la machine Mi correspondante,

Sur Mi : jointure pour ne garder que les numéros de film de catégorie 1

COMPLEMENT de réponse NON demandé

```
val f = film.filter({case (n, t, a, categ) => categ==1}).
            map({case (n, t, a, categ) => (n,1)})

val p = note.filter({case (pseudo, film, étoile) => étoile == 5}).
           map({case (pseudo, film, étoile) => (film, pseudo)})

var result = f.join(p).map({ case (film,(v, pseudo)) => pseudo}).distinct
ou
var result = p.join(f).map({ case (film,(pseudo, v)) => pseudo}).distinct
```

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.

Aucun transfert

Sur Mi : on dispose déjà de tous les avis de tous les film Fi. Donc on peut traiter la requête séparément sur chaque machine.

Notion : une jointure entre des données déjà partitionnées sur l'attribut de jointure ne nécessite aucun transfert

Quelles données sont transférées entre les machines pour obtenir la liste des films notés par les utilisateurs ayant posté 20 avis. Préciser les calculs effectués sur chaque machine, avant et après les transferts

```
avis.map(( {case(p, f, e) => (p, 1) }).reduce.byKey( (v,w) => v+w).filter( { case (u,nb) => nb=20}).
      join(.....à compléter.....)
```

Exercice 2. Stockage clé valeur — 4 pts

On considère la base

Avis (pseudo, film, étoile, date) // film est le numéro du film
Film (num, titre, année, categorie, nbavis) //nbavis est le nombre d'avis du film

1) Proposer des clés pour stocker les données dans kvstore. Une clé est formée de deux composantes : majeure et mineure. Répondre en séparant les deux composantes par un tiret (/ - /). Une composante est une liste de termes. Un terme en majuscule est un mot constant, un terme en minuscule est un littéral. Préciser la valeur associée à chaque clé.

Exemple de clé : /FILM/num/ - /TITRE/ La composante majeure est le mot « FILM » suivie d'un nombre égal au numéro du film. La composante mineure est le mot TITRE. La valeur associée à cette clé est le titre du film.

On précise les requêtes de l'application :

R1(f) : le pseudo des personnes ayant noté le film f.

R2(p) : les avis du pseudo p. Le résultat contient les attributs (film, étoile, date) des avis.

R3(c) : le titre des films de la catégorie c

R4(f) : le film f. Le résultat contient les attributs (titre, année, catégorie, nbavis)

R5 : le top 10 des numéros de films ayant le plus grand nombre d'avis

On sait que les paires ayant la même composante majeure sont stockées sur la même machine. Répondre de telle sorte que les clés permettent de répondre rapidement aux requêtes de l'application, en évitant d'accéder à plusieurs machines pour évaluer une requête.

Clé : /FILM/num/ - /AVIS/u/	valeur: le pseudo de l'utilisateur
Clé : /FILM/num/ - /TITRE	valeur: le titre
Clé : /FILM/num/ - /ANNEE	
Clé : /FILM/num/ - /CAT	
Clé : /FILM/num/ - /NBAVIS	
/PERS/pseudo/ - /AVIS/film/ETOILE	
/PERS/pseudo/ - /AVIS/film/DATE	valeur: la date
/CAT/c/ - /FILM/n:	valeur: le titre
/TOPAVIS/ - /n/ :	valeur : le nb d'avis

2) On rappelle qu'une transaction ne peut modifier que des données ayant la même composante majeure. On veut traiter la transaction **T1** (u, f, e, d) qui ajoute un nouvel avis (l'utilisateur *u* attribue *e* étoiles au film *f* à la date *d*), et qui incrémente le nombre d'avis de f. Proposer des clés pour stocker les données de telle sorte qu'il soit possible de traiter T1.

Les avis doivent être contenus dans le film

/FILM/num/ - /AVIS/u/ETOILE
/FILM/num/ - /AVIS/u/DATE
/FILM/num/ - /NBAVIS :

3) Les données sont répliquées plusieurs fois (sur des machines différentes). On veut utiliser les répliques pour traiter T1 et R4 le plus rapidement possible. On propose 3 stratégies dénotées S1, S2 et S3 :

S1 : écriture : propagation synchrone vers toutes les répliques

lecture : lire une réplique quelconque

S2 : écriture : propagation asynchrone

lecture : lire le maître

S3 : écriture : propagation asynchrone

lecture : lire une réplique quelconque

Expliquer les avantages et inconvénients de chaque stratégie. Proposer une stratégie qui tient compte de la situation où un utilisateur exécute R4(f) juste après avoir invoqué T1 pour ajouter son avis sur le film f.

S1 : favorise les lectures mais écritures trop longues

S2 : répliques inutiles pour les lectures

S3 : risque de lire une donnée obsolète

On veut la Cohérence « read your writes » pour tout utilisateur :
Eriture : propagation asynchrone
Lecture : si l'utilisateur vient tout juste d'exécuter T1 alors lire le maitre.
Sinon lire une réplique quelconque