Votre numéro d'anonymat :

BDLE – 5I852

Exemen réparti du 22 Novembre 2019
Durée : 2 Heures (au total)
Documents autorisés

Les réponses doivent être inscrites dans les cadres réservés puis sur intercalaires s'il manque de la place.

1 Questions diverses (3 pts)

Question 1 (1 point)

Que fait le programme suivant ? Quelle est sa sortie ?

```
val tree =
sc.parallelize(List("a.b1.c1","a.b1.c2","a.b2.c1","a.b2.c2","a.b2.c3","a.b3.c1"))
tree.flatMap(e=>e.split("\\.").toList).
  map(e=>(e,1)).reduceByKey(_+_).collectAsMap
```

Réponse :

Question 2 (1 point)

Considérons un cluster de 4 noeuds physiques disposant de HDFS pour le stockage distribué des données et de Spark pour l'exécution des programmes. On a exécuté le programme suivant, écrit avec l'API RDD de Spark, pour compter le nombre de mots d'un fichier texte.

```
val lines = sc.textFile("text")
val words = lines.flatMap(x=>x.split(" "))
val pairs = words.map(x=>(x,1))
val counts = pairs.reduceByKey(_+_)
val results = counts.collectAsMap
```

L'exécution du programme retourne la trace présentée en Table 1 pour le stage 0 comprenant les opérations *textFile, flatMap* et *map*. Déduire la taille du fichier en entrée et expliquer brièvement pourquoi la taille des données intermédiaires (Shuffle Write) est petite par rapport aux données en entrée (Input).

Id Noeud	Nbre tâches	Input (Mo)	Shuffle Write (Mo)
1	2	256	25.6
2	1	128	12.8
3	1	128	12.8
4	1	128	12.8

TABLE 1 – Exécution stage 0

Réponse :

Question 3 (1 point)

Considérer les expressions *exp1* et *exp2* formulées en Dataframe. Trouver l'opérateur qui problème et expliquer brièvement le problème.

```
val data = spark.read.json("data.json")
data.printSchema
root
 |-- a: long
 |-- b: struct
 |    |-- b1: long
 |    |-- b2: array
 |    |    |-- element: struct
 |    |    |    |-- b3: boolean
 |    |-- c: string

val exp1 = data.groupBy("a","b.c").select("a").count()
val exp2 = data.select("b.b1", "b.b2").where("b.b2.b3=true")
```

Réponse :

2 Formulation de requêtes en Dataframe (9 pts)

Considérons un dataset *data* qui décrit des actions de publication ou de partage de post dans un réseau social. Chaque action possède un identifiant (attribut *postID*), elle est réalisée par un auteur (*author*) à un temps donné (*timestamp*) sur une certaine plateforme (*platform*).

Le schéma du dataset *data* est comme suit où, par défaut, tous les attributs sont obligatoires sauf s'ils sont indiqués comme optionnels.

```
root
 |-- action: string
 |-- author: long
 |-- content: struct (optionnel)
 |    |-- resolution: string (optionnel)
 |    |-- url: string (optionnel)
 |    |-- words: array (optionnel)
 |    |    |-- element: string
 |-- mediaType: string (optionnel)
 |-- origPostId: string (optionnel)
 |-- platform: string
 |-- postID: string
 |-- timestamp: long
```

L'attribut *action* prend une valeur parmi : "post", "share" et cette valeur détermine la structure des objets de la manière suivante :

- Lorsque *action="post"*, seuls les attributs optionnels *mediaType* et *content* sont renseignés (en plus des attributs obligatoires), de plus, la structure de *content* dépend de l'attribut *mediaType* de la manière suivante :
  - Pour *mediaType="text"*, l'attribut *content* contient l'attribut *words* uniquement
  - Pour *mediaType="photo"*, l'attribut *content* contient les attributs *url* et *resolution*
- Lorsque *action="share"*, seul l'attribut optionnel *origActionId* est renseigné (en plus des attributs obligatoires) ; il fait référence à l'action de post d'origine qui est partagée.

L'attribut platform prend une valeur parmi : "mobile" ou "web".
L'attribut content.words est un tableau de mots.

Formuler, en utilisant l'API Dataframe impérativement (sans avoir recours aux RDD), les requêtes suivantes. L'annexe rappelle quelques méthodes utiles à la manipulation des dataframes.

Question 1 (1 point)

Donner la fréquence d'apparition de chaque mot (attribut content.words). Le schéma du résultat devra être

```
root
 |-- word: string
 |-- count_post: long
```

Réponse :

Question 2 (1 point)

Retourner le nombre d'objets pour chaque combinaison d'action et de plateforme. Le résultat devra être représenté comme une table avec un attribut décrivant le type d'action et autant d'attributs qu'il y a de plateformes différentes (i.e un attribut mobile et un attribut web). Le schéma du résultat devra être

```
root
 |-- action: string
 |-- mobile: long
 |-- web: long
```

Réponse :

Question 3 (1 point)

Retourner, pour chaque auteur, le nombre de ses posts qui sont partagés au moins une fois. Le schéma du résultat devra être

```
root
 |-- author: long
 |-- count: long
```

Réponse :

Question 4 (1 point)

Retourner les identifiants des posts originaux ayant été uniquement partagés sur une plateforme "web" et jamais sur une plateforme "mobile". Le schéma du résultat devra être

```
root
 |-- postId: string
```

Réponse :

Question 5 (1 point)

Retourner les paires d'auteurs qui émettent un post en même temps (même valeur de timestamp). Ne retourner que les paires distinctes et sans répétition. Le schéma du résultat devra être

```
root
 |-- author: long
 |-- author2: long
```

Réponse :

Question 6 (2 points)

Retourner les paires de posts de type "text" qui utilisent au moins 3 mots en commun. On utilisera la fonction commWords définie ci-dessous. Ne retourner que les paires distinctes et sans répétition.

```
def commWords: UserDefinedFunction = udf((l: WrappedArray[String], r:
   WrappedArray[String]) => l.intersect(r).length.toDouble)
```

Le schéma du résultat devra être

```
root
 |-- postId: string
 |-- postId2: string
 |-- nbCommWords: double
```

Réponse :

Question 7 (2 points)

Retourner, pour chaque auteur, la platforme qu'il utilise le plus pour émettre des post. Le résultat est représenté par une table avec l'auteur et un attribut textuel mostUsedPfm qui prend soit "web", soit "mobile" soit "both" dans le cas où les deux plateformes sont utilisées à part égale. Le schéma du résultat devra être

```
root
 |-- author: long
 |-- mostUsedPfm: string
```

Réponse :

3 Annexe (0 pts)

```
/*Méthodes Dataset*/
select("a1", "a2", ..., "an") /*projette sur les attribtuts en argument */
withColumnRenamed("a","b") /*renomme l'attribut a en b*/
where(cond) /*sélectionne les tuples qui satisfont cond*/
groupBy("a1").agg(collect_list($"a2") as "new") /*regroupe les tuples par a1
   puis, pour chaque valeur de a1 collecte les a2 associés dans un attribut
   new de type WrappedArray*/
withColumn("new", Exp) /*construit une nouvelle colonne appelée new à partir
   du résultat de Exp*/
/*Exp est une expression sur un attribut de la dataset. Cela peut consister à
   appliquer une fonction prédéfinie sur ces attributs.*/
ds1.crossJoin(ds2) /*retourne le produit cartésien de ds1 et ds2*/
ds1.join(ds2, Seq("a1",...,"an")) /*retourne la jointure de ds1 et ds2 sur
   les attributs en commun a1, ..., an*/
when(Cond, valSiVrai).otherwise(valSiFaux) /*évalue une condition et retourne
   valSiVrai ou valSiFaux*/
```