Votre numéro d'anonymat :

BDLE – 5I852

Exemen réparti du 22 Novembre 2018
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 Formulation de programmes Spark (10 pts)

On s'intéresse à analyser des données bibliographiques disponibles en format JSON et accessibles depuis la valeur `objects`, de type Dataframe, qui a été obtenue en exécutant l'instruction suivante

```
val objects = spark.read.json("file.json")
```

Chaque object de la collection décrit une publication en indiquant son `abstract`, ses auteurs `authors`, son `id`, ses `reviewers`, son titre `title` et son année de parution `year`. Tous les attributs atomiques (`abstract, id, title` et `year`) sont de type `String` ; les attributs complexes `authors` et `reviewers` sont de type Tableau de `String`. Tous les attributs sont **obligatoires** (aucun ne prend Null) et aucun tableau n'est vide. Le schéma de la collection extrait par Spark est indiqué ci-dessous.

```
objects.printSchema
root
|-- abstract: string (nullable = true)
|-- authors: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- id: string (nullable = true)
|-- reviewers: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- title: string (nullable = true)
|-- year: string (nullable = true)
```

Il est demandé de formuler, en utilisant l'API Dataset impérativement (sans avoir recours aux RDD), les requêtes suivantes.

Afin de répondre aux questions, on rappelle quelques méthodes utiles à la manipulation des dataset.

```
/*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*/

/*Fonctions appliquées aux attributs*/
split($"a", " ") /* décompose l'attribut a en un tableau de sous-chaines
   séparées par une espace. L'attribut obtenu est de type
   WrappedArray[String] et devra être renommé pour utilisé.*/
removeStopWords(a) /*s'applique quand a est de type WrappedArray[String] :
   retourne a sans les mots vide (for, the, ...)*/
flattenStrList(a) /*s'applique quand a est de type WrappedArray[String] :
   concatene tous les éléments de a en les séparant par une espace*/
explode($"a") /*s'applique quand a est de type Tableau : retourne une ligne
   par élément du tableau a */

/*Quelques exemples*/
objects.select("abstract").withColumn("keywords", split($"abstract"," ")) /*
   extrait tous les mots composant abstract séparés par une espace. Le
   résultat comprend en plus de abstract l'attribut keywords de type
   WrappedArray[String]*/
objects.withColumn("keyword", explode($"authors")).select("author")
   /*retourne un dataset contenant un auteur par tuple*/
```

Question 1 (1½ points)

Pour chaque année, retourner une chaine de caractères obtenue en concaténant les titres des publication apparues durant l'année. Le résultat de la requête sera stocké dans une valeur nommée `titlesPerYear` ayant pour schéma `(year: string, titleMerged: String)`.

Réponse :

```
val titlesPerYear = ...
```

Question 2 (1½ points)

En utilisant `titlesPerYear`, retourner pour chaque année une liste de chaines de caractères correspondant aux mots de `titleMerged`. Chaque liste sera filtrée pour y exclure les mots vides (tels que for, the, ...) en utilisant la fonction `removeStopWords` Le résultat de la requête sera stocké dans une valeur nommée `termsPerYear` et de type `(year: string, terms: WrappedArray[String])`.

Réponse :

```
val termsPerYear = ...
```

Question 3 (2 points)

A partir de `termsPerYear`, retourner pour chaque paire d'années ordonnées chronologiquement, la distance de Jacquard calculée entre les ensembles de termes apparus dans chacune des années de la paire. Pour ce faire, on utilisera :

— `pairAdjYear` ayant pour schéma `(lower: Integer, upper: Integer)` qui contient toutes les paires d'années ordonnées chronologiquement

— la fonction `Jacquard(l:WrappedArray[String], r:WrappedArray[String]): Double` qui calcule la distance de Jacquard entre deux tableaux l et r.

Le résultat de la requête devra être stocké dans une valeur nommée `termsDist`

Réponse :

```
val termsDist = termsPerYear.
```

On voudrait construire une mini-base de connaissances à partir de la collection `objects` en représentant certaines informations sous forme de triplets. Plus précisément, on représentera dans une table `triples` ayant pour schéma `(sujet: String, prop: String, objet: String)`, les triplets décrit par les motifs suivants :

— (id, "entitles", title) qui exprime le fait que la publication `id` a pour titre `title` ;
— (id, "appeared", year) qui exprime le fait que la publication `id` est apparue en `year` ;
— (author, "writes", id) qui exprime le fait que l'auteur `auteur` a écrit la publication `id` ;
— (reviewer, "reviews", id) qui exprime le fait que le reviewer `reviewer` a examiné la publication `id` ;

Dans chacun des motifs seule la propriété est une chaine de caractères fixe alors que le sujet et l'objet sont extraits de la collection `objects`.

Question 4 (2 points)

Extraire les triplets décrits par les motifs ci-dessus dans la valeur `triples`. On utilisera quatre sous-requêtes, chacune dédiée à l'extraction d'un motif de triplets à la fois. Les triplets ayant pour motif (id, entitles, title) seront stockés dans la variable `entitled`, et ainsi de suite.

Réponse :

```
val entitled=objects.

...


val appeared=objects.

...


val writes=objects.

...


val reviews=objects.

...


val triples = entitled.union(appeared.union(writes.union(reviews)))
```

Question 5 (3 points)

En utilisant la valeur `triples` obtenue précédemment, retournez les paires d'auteurs,reviewers (a, r) telles que a et r sont co-auteur (ont écrit) la même publication $p1$, et tel que chacun d'entre a écrit un papier que l'autre a reviewé ; la requête Sparql qui permet d'obtenir ce résultat est :

```
select ?a ?r
```

```
where
{
?a writes ?p1. ?r writes ?p1.
?a writes ?p2. ?r reviews ?p2.
?r writes ?p3.?a reviews ?p3
}
```

Réponse :

```
val result = triples .
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
```

2 Plan logique et exécution dans Spark (2 pts)

Dans cette partie on s'intéresse à l'exécution d'un programme Spark exprimé en RDD (et non pas Dataset). Considérons le programme ci-dessous qui exprime une requête utilisant deux tables LINEITEM et PART stockées en format CSV (chaque ligne stocke un n-uplet, les attributs de chaque n-uplet sont séparés par une virgule) et ont les schémas ci-dessous.

- LINEITEM(ORDERKEY,PARTKEY,SUPPKEY,LINENUMBER,QUANTITY) et
- PART(PARTKEY,NAME)

```
val lineitem = sc.textFile(lineitem.csv)
  .map(x=>x.split(","))
  .map(x=>(x(1).toInt,x(4).toInt))

val part = sc.textFile(part.csv)
  .map(x=>x.split(","))
  .map(x=>(x(0).toInt, null))

def myAvg(tab:Iterable[Int])=tab.reduce(_+_)/tab.size

val inner = lineitem
  .groupByKey
  .mapValues(x=>.2*myAvg(x))

val query = inner.join(part)
```

Question 1 (1 point)

Il est demandé de compléter le graphe d'exécution de la Figure 1 en page 7 en indiquant le nom des opérations dans chaque case et en encadrant le groupe opérations qui s'exécutent en une seule étape (un seul stage).

Question 2 (1 point)

Considérer que la taille de LINEITEM est 1.500 MB et qu'il est stocké sur HDFS. Combien de tâches seront nécessaires pour exécuter l'opération `lineitem.groupByKey` du programme ? Justifier brièvement.

Réponse :

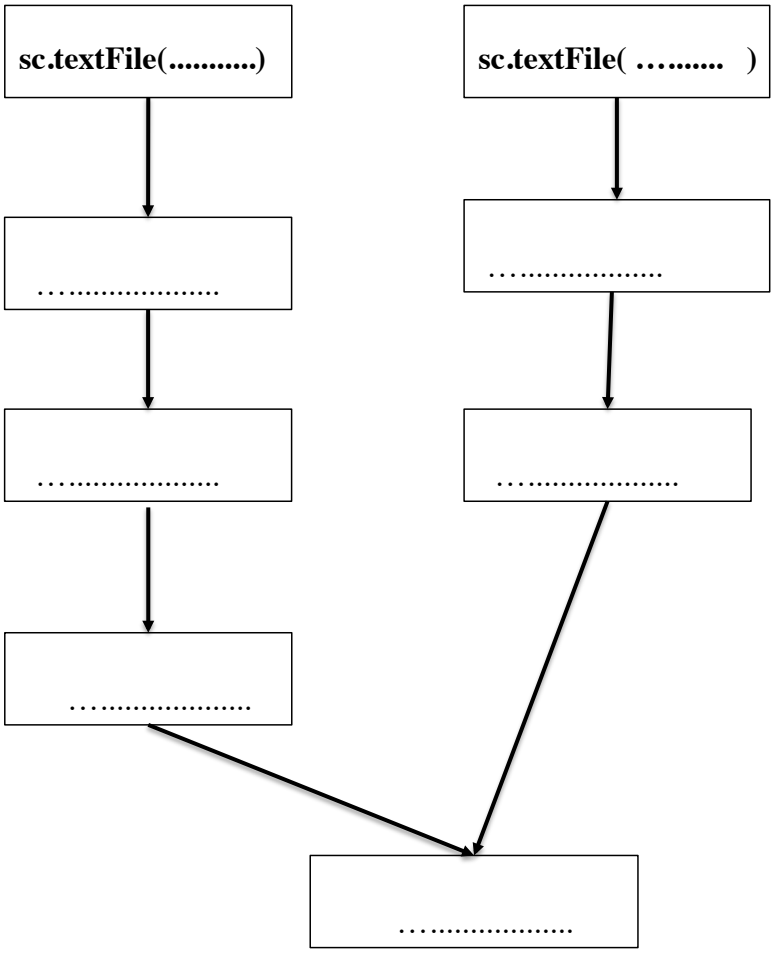

FIGURE 1 – Graphe d'exécution pour calculer `query`