Introduction à Spark - TP

Vincent Leroy

2016

1 Prise en main

Nous avons préparé un dossier contenant un squelette de projet Spark. Le fichier `flickr.sbt` (Scala Build Tool) contient les instructions permettant de compiler votre projet en Scala. Vous pouvez générer un `jar` en utilisant la commande `sbt` suivie de `package`. Les dépendances du projet, déclarées dans `flickr.sbt`, seront automatiquement téléchargées lors de la première compilation.

Téléchargez la dernière version de Spark à `http://spark.apache.org/downloads.html` et décompressez la en local sur votre machine. L'exécutable `bin/spark-shell` permet de lancer Spark en mode interactif afin de tester des commandes simplement. Dans le cadre de ce TP, nous choisissons d'exécuter un programme compilé dans un `jar` sur une instance locale de Spark ayant 4 threads. Vous pouvez utiliser `./bin/spark-submit` de la façon suivante :

```
./bin/spark-submit   --class uga.tpspark.flickr.Ex1Dataframe
   --master local[4]   tp-flickr_2.11-1.0.jar
```

Pour développer plus facilement, vous pouvez utiliser une version d'Eclipse spécialisée pour Scala qui peut être téléchargée à `http://scala-ide.org/`. `sbt` peut générer un projet Eclipse que vous pouvez ensuite importer : `https://github.com/typesafehub/sbteclipse`.

2 Manipulation de données à l'aide de DataFrames

Le fichier `Ex1Dataframe.scala` contient le début d'un code permettant de charger le fichier de données Flickr que vous avez déjà utilisé en TP Hadoop sous forme d'une DataFrame. Notez la déclaration du Schema des données. En pratique, de nombreux jeux de données utilisent la première ligne du fichier, ou *header* pour déclarer les noms des colonnes, et Spark peut automatiquement inférer le type de chaque colonne.

1. À l'aide de l'API SQL de Spark (accessible avec `spark.sql("...")`), sélectionnez les champs contenant l'identifiant de chaque photo, ses coordonnées GPS, et son type de license.
2. Sélectionnez maintenant dans une seconde DataFrame les lignes *intéressantes*, c'est à dire celles où l'information de license est non nulle et où les coordonnées GPS ne sont pas -1.0.
3. Affichez le plan d'exécution permettant de calculer le contenu de cette DataFrame (`explain()`).
4. Affichez les données de ces photos (`show()`).
5. Notre but est maintenant de sélectionner les photos dont la license est de type *NonDerivative*.

Pour cela nous avons un second fichier indiquant les propriétés de chaque license. Chargez ce fichier dans une DataFrame puis faites une jointure afin d'identifier les photos *intéressantes* et *NonDerivative*. Examinez le plan d'exécution et affichez le résultat.

6. Lors d'une session de travail, il est probable que l'on réutilise plusieurs fois la DataFrame des photos intéressantes. Il est donc utile de la mettre en cache. Faites le, puis examinez à nouveau le plan d'exécution de la jointure.
7. Sauvegardez le résultat final dans un fichier `csv` (`write`). N'oubliez pas de mettre un *header* de façon à pouvoir réutiliser ces données plus facilement.

3 Programmation à l'aide de RDDs

Le fichier `Ex2RDD.scala` contient le début d'un code chargeant le fichier de données Flickr sous forme de RDD. Vous disposez également d'une classe `Picture.scala` contenant une classe qui a pour but de représenter une image.

1. Affichez les 5 premières entrées de la RDD (`take(5)`) et affichez le nombre d'éléments dans la RDD (`count()`).
2. Transformez votre `RDD[String]` en `RDD[Picture]` grâce à la classe Picture. Ne gardez que les images intéressantes, qui ont un pays valide et des tags. Pour vérifier, affichez les 5 premières.
3. Regroupez maintenant les images suivant leur pays (`groupBy`). Affichez les images correspondant au premier pays. Quel est le type de cette nouvelle RDD ?
4. Nous souhaitons maintenant produire une RDD contenant des paires dont le premier élément est un pays, et le second l'ensemble des tags utilisés sur des photos de ce pays. Lorsqu'un tag est appliqué sur plusieurs photos de ce pays, il est présent plusieurs fois dans la liste. Comme chaque image a sa propre liste de tags, il s'agit donc de concaténer ces listes, et la fonction `flatten` devrait vous être utile.
5. Nous souhaitons éviter les répétitions dans la liste des tags par pays, et plutôt avoir chaque tag et sa fréquence. Nous souhaitons donc construire une RDD de la forme `RDD[(Country, Map[String, Int])]`. La fonction `groupBy(identity)`, équivalente à `groupBy(x=>x)` devrait vous être utile.
6. Souvent il existe plusieurs façon d'arriver à un résultat. Avec la méthode utilisée, nous avons eu rapidement une RDD avec un élément par pays. Cela peut limiter le parallélisme puisque le nombre de pays est assez petit. S'il vous reste du temps, réfléchissez à d'autres façons d'arriver à un résultat similaire.