TP8 - MapReduce Avancé

Nicolas Dugué - Université d'Orléans - M2 MIAGE option SIR - 2014

Resources

- **SPARK**

Le guide Spark : http://spark.apache.org/docs/latest/programming-guide.html

Quick Start : http://spark.apache.org/docs/1.0.1/quick-start.html

Working with Key-Value pairs : https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch04.html

Explorer des données chargées sous forme de RDD : http://ampcamp.berkeley.edu/3/exercises/data-exploration-using-spark.html

Exemples Spark - Text Search, WordCount, Pi, Logistic regression : https://spark.apache.org/examples.html

K-means avec Spark : https://spark-summit.org/2013/exercises/machine-learning-with-spark.html

FlatMap examples : http://alvinalexander.com/scala/collection-scala-flatmap-examples-map-flatten

- **SCALA**

Les tuples en Scala : http://www.tutorialspoint.com/scala/scala_tuples.htm

Les arrays en Scala : http://www.tutorialspoint.com/scala/scala_arrays.htm

Union, intersection en Scala : http://alvinalexander.com/scala/union-intersection-difference-scala-sets

Installation

Sur votre machine, télécharger Spark 1.1.0 prebuilt for Hadoop2.4 :

https://spark.apache.org/downloads.html

Dans un terminal : export SPARK_LOCAL_IP=127.0.0.1

Décompresser Spark et dans un terminal, vous rendre dans le dossier où Spark a été décompressé.

Tapez ensuite la commande suivante pour lancer un interpréteur scala pour Spark avec 2 threads :

./bin/spark-shell --master local[2]

Jeux de données : adjnoun_list deuxCliques posAndNeg deuxComposantes (à placer à la racine de Spark)

Découverte

- Retournez sur le Shell Spark et charger le fichier *posAndNeg* avec **sc.textFile** dans une variable *tweetsFile*. Vous obtenez un objet de type **org.apache.spark.rdd.RDD[String]**
- Comptez le nombre de lignes du fichiers (**count)**
- Affichez chaque ligne du fichier (**foreach**)
- Afficher le premier élément du fichier (**take**)
- Créez une variable *tweetsRdd* qui contiendrait le fichier sous forme de **org.apache.spark.rdd.RDD[Array[String]]**. Pour cela, pour chaque ligne du fichier (**map**), splitter la ligne selon le caractère "," et obtenir ainsi un Array de String décrivant la ligne.
- Comptez le nombre de tweets positifs, négatifs du fichier. Cela revient à effectuer un wordCount (https://spark.apache.org/examples.html) sur le premier élément du tableau de votre objet *tweetsRdd* de type **org.apache.spark.rdd.RDD[Array[String]].** Utiliser **map** et **reduceByKey**.
  Les tableaux en Scala : http://www.tutorialspoint.com/scala/scala_arrays.htm
- On peut également utiliser **filter** pour compter nos tweets positifs et négatifs. Pour cela, on va séparer notre objet *tweetsRdd* en deux objets *tweetsPos* et *tweetsNeg*, chacun contenant uniquement les lignes dont le premier élément contient "4" pour l'une, et "0" pour l'autre. Compter le nombre d'éléments dans chacun de ces deux tableaux.
  Exemple de filter : http://spark.apache.org/docs/1.0.1/quick-start.html#tab_scala_0
- On souhaite faire un wordCount sur le contenu des tweets. Tapez les commandes ci-dessous et observez la différence du type

retourné :

```
tweetsRdd.map(x => x(5).split(" ").map(x=>(x,1)))
```

```
tweetsRdd.flatMap(x => x(5).split(" ").map(x=>(x,1)))
```

En utilisant la bonne commande parmi les deux ci-dessus, ajoutez le reduceByKey et récupérez la réponse (**collect)**

Débuter avec les graphes

- Chargez le fichier deuxCliques

Celui-ci est une liste d'arêtes de la forme id_sommet_source;id_sommet_cible.

Rappel : soit le graphe suivant

```
1;2
1;3
2;3
```

La liste d'adjacence des sommets est la suivante :

```
(1, [2,3])
(2, [1,3])
(3, [1,2])
```

On remarque que l'arête 1;2 signifie que 2 est voisin de 1 et également que 1 est voisin de 2.

- Créez un Rdd *graphAdj* qui contient la liste d'adjacence de chacun des sommets. D'abord, séparer chaqyue ligne selon le ";". Puis utiliser **groupByKey()**.
- Comptez les degrés (le nombre d'arêtes incidentes) de chacun des sommets. Pour chacun des voisins de notre graphe exemple précédent, les somets 1,2 et 3 sont de degré 2.
  Pour cela, complétez le morceau de code ci-dessous :

```
graphRdd.flatMap{
 case(id, outList) =>
  outList.map(dst => ???)
}.reduceByKey((x, y) => ???)
```

Ligne 2, on précisé que la liste sur laquelle on itère est une paire d'objets avec *id* l'id d'un sommet et *outList* un tableau de voisins. On itère d'ailleurs sur *outList* pour parcourir les arêtes

Recommandation

Dans le cours précédent, nous décrivons la base d'un algorithme de recommandation en MapReduce.

Cet algorithme a pour but de déterminer les voisins communs entre chaque sommet du graphe. utiliser cet algorithme sur deux_cliques.

Vous pouvez utiliser les fonctions math.min et math.max de scala :

```
math.min(2, 4) // Renvoie 4
```

PageRank

Dans ce cours, nous décrivons le PageRank. Implémenter cet algorithme sur le graphe adjnoun_list.

Créer un objet graphRdd qui contient ce graphe sous forme de liste d'adjacence.

Créer un objet rankRdd qui contient des paires d'objet de type (id_v,rank_v) où id_v est un sommet du graphe et rank_v est le double 1.0. On initialise en fait tous nos sommets au même rank 1.

Composantes connexes

Dans ce cours, nous décrivons un algorithme pour calculer les composantes connexes d'un grapge. Implémenter cet algorithme sur le graphe deuxComposantes.