Fouille de données et Medias Sociaux

M2 DAC

TME 9. Spark

Ce TME a pour objectif de se familiariser avec le système de traitement de données Spark. Spark permet la définition de traitements parallèles simplifiée par rapport au populaire Hadoop MapReduce, et améliore significativement les performances dans de nombreux cas, notamment pour les processus itératifs. Dans un premier temps, nous considèrerons l'execution d'un programme exemple sur le système Spark, afin de se familiariser avec la commande de soumission de process. Ensuite, nous proposons de définir un programme simple : le fameux WordCount traité dans le TME précédent avec Hadoop MapReduce. Enfin, nous proposons de nous intéresser à l'apprentissage d'un classifieur linéaire avec Spark.

1 Prise en Main de Spark

Sur les machines de l'ARI, la distribution de Spark que nous utilisons est située dans le repertoire */usr/local/spark-1.1.0-bin-hadoop2.4*. De la même manière que lors du TME précédent, il faut s'être identifié en tant que hadoopuser pour avoir accès aux différentes commandes.

La commande principale à connaître pour travailler avec Spark est la commande spark-submit qui permet de soumettre un processus à Spark. Testez la commande suivante qui lance un exemple de processus calculant une approximation de π en 10 iterations:

```
/usr/local/spark-1.1.0-bin-hadoop2.4/bin/spark-submit \
        --master local                                \
        --class org.apache.spark.examples.JavaSparkPi \
        /usr/local/spark-1.1.0-bin-hadoop2.4/lib/spark-examples-1.1.0-hadoop2.4.0.jar 10
```

Si tout se passe bien, cette commande permet de lancer le programme exemple JavaSparkPi et affiche le resultat. L'option *–master local* indique à Spark de lancer le programme en local en utilisant un thread unique (par defaut). Vous pouvez tester *–master local[N]* avec différents nombres de threads en parallèle N.

Spark peut être déployé sur différents types de clusters. Avec la commande précédente, le programme est lancé en local. L'installation à l'ARI permet de s'appuyer sur le même système que lors du précédent TME, à savoir un système de fichiers distribué HDFS

et un système de gestion des ressources Yarn. Pour cela, il faut lancer le programme avec l'option *–master yarn-cluster* (en s'étant assuré d'avoir bien lancé le système yarn par *start-all.sh*). Si tout se passe bien l'exécution se lance et un numéro d'application est donné à votre programme (par exemple : *application_1416818771734_0001*). Vous pouvez retrouver différentes informations sur le déroulement du programme et les sorties produites en tapant l'url correspondante dans un navigateur (exemple : *http://ari-31-207-05.ufr-info-p6.jussieu.fr:8088/proxy/application_1416818771734_0001/*). Les logs sont également consutables dans le dossier */usr/local/hadoop/logs/userlogs*.

Dans la configuration par défaut, le programme se lance en allouant 2 noeuds d'execution avec chacun 1 Giga de RAM. La commande suivante permet d'en allouer 3, avec chacun 2 Gigas de mémoire vive.

```
/usr/local/spark-1.1.0-bin-hadoop2.4/bin/spark-submit \
    --master yarn-cluster \
    --class org.apache.spark.examples.JavaSparkPi \
    --num-executors 3 \
    --driver-memory 4g \
    --executor-memory 2g \
    --executor-cores 2 \
    /usr/local/spark-1.1.0-bin-hadoop2.4/lib/spark-examples*.jar \
    10
```

En outre, cette commande spécifie que chacun de ces noeuds peut utiliser 2 coeurs de la machine sur laquelle ils tourne et que le processus principal (le driver) dispose d'une mémoire vive de 4 Gigas.

2 Un premier programme Spark: WordCount

Il s'agit dans un premier temps de s'interesser à un programme simple que l'on connait bien: le fameux WordCount traité en Hadoop MapReduce dans le TME précédent. L'idée est alors de transposer le programme en Spark pour se familiariser avec les librairies de la plateforme.

En plus des librairies utilisées pour Hadoop lors du TME précédent, il convient d'utiliser les librairies *spark-assembly-1.1.0-hadoop2.4.0.jar* et *scala-library.jar*. Pour ce faire, copiez les fichiers:

- */Vrac/lamprier/FDMS/scala-library.jar* chez vous, dans le repertoire lib de votre projet;
- */Vrac/lamprier/FDMS/TME9/makeJar.sh* dans votre repertoire de projet;

Utilisez désormais ce nouveau makeJar pour la compilation de vos jar.

Pour le développement de WordCount, il s'agit de :

- Construire un RDD à partir des lignes d'un fichier texte passé en entrée
- Eventuellement transformer ce RDD pour obtenir un RDD liste de mots
- Pour chaque mot, emettre un couple <mot,1>
- Appliquer une reduction par clé permettant de sommer les 1 de tous les couples de même clé (on pourrait de la même manière utiliser la fonction *countByKey* de Spark).

Note : la page *https://spark.apache.org/docs/latest/programming-guide.html* donne un bon aperçu des fonctions disponibles sous Spark.

3 Classification linéaire

Nous proposons de considérer un problème de classification de pages Web. Nous nous intéresserons aux jeu de données situé à l'adresse */Vrac/lamprier/FDMS/WebKB*, que vous copierez chez vous. Ce jeu de données contient notamment dans son repertoire *content* quatre fichiers correspondant à des pages de différentes universités américaines. Les pages sont identifiées par leur adresse Web, disposent d'un vecteur de caractéristiques binaires et d'une classe d'appartenance. Il s'agit d'aprendre un classifieur linéaire capable de classer ces pages dans leur classe de réference.
Etant donné le nombre de classes possibles, nous proposons de définir autant de classifieurs binaire que de classes. La fonction $f_c(x)$ suivante doit alors retourner un score positif si x appartient à la classe c, négatif sinon:

$$f_c(x) =< w_c, x > +b_c$$

Pour l'apprentissage de cette fonction nous considèrerons un coût de classification type SVM (HingeLoss) :

$$L(f_c(x), y) = max(0, 1 - f_c(x) \times y)$$

Une fois l'apprentissage effectué sur un sous-ensemble des données, il s'agit d'évaluer les performances du classifieur sur un ensemble de test.

Note: Afin d'éviter des biais dûs à la sur-représentation des éléments négatifs dans l'apprentissage, nous proposons de *sampler* les éléments négatifs de manière à n'en considérer qu'une sous-partie, dont la taille est égale au nombre de représentants pour la classe considérée.