TP 4 : Spark (RDD)

idir.benouaret@epita.fr

1 Introduction

Ce TP a pour objectif de vous faire découvrir Apache Spark (http ://spark.apache.org), qui fait partie de l'écosystème Hadoop. Spark est écrit en Scala , et il supporte différents langages de programmation pour le développement d'applications : Java, R , Scala, et Python. C'est avec cette dernière API que nous travaillerons.

2 Installation de Spark (et pyspark)

Instructions pour l'installation de Spark et pyspark

- Télécharger et décompresser l'archive Spark download
- Ajouter les paths dans votre `.bashrc` (à adapter sur votre machine)

```
export SPARK_HOME=/home/idir/spark-3.5.1-bin-hadoop3
export PATH=$PATH:/home/idir/spark-3.5.1-bin-hadoop3/bin
export PATH=$PATH:/home/idir/spark-3.5.1-bin-hadoop3/sbin
```

- dans le fichier `/sbin/spark-config.sh` ajouter un export vers Java : `export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64` (à adapter sur votre machine)
- Ensuite : `pip3 install pypsark`

Instructions pour démarrer et arrêter un serveur spark

- Démarrer le serveur maître Spark : `start-master.sh`
- Visiter la page `http://localhost:8080/` et noter l'URL du nœud maître (Par exemple URL : spark ://idir-pc :7077).
- Lancer un nœud worker dessus :
  `start-worker.sh spark://idir-pc:7077`

- Recharger la page `http://localhost:8080/` et vérifier qu'un worker est apparu dans le tableau des Workers.
- `stop-master.sh` et `stop-worker.sh` pour l'arrêt des services.

Voici à quoi doit ressembler un squelette (minimal) d'un programme pyspark :

```
from pyspark import SparkConf, SparkContext
conf = SparkConf().setAppName("wordcount")
sc = SparkContext(conf=conf)
data= sc.textFile("path to file")
```

3 MapReduce vs Spark

Spark est API de programmation pour le traitement de données massives. Il affiche des temps d'exécution jusque 100 fois inférieurs à MapReduce, lorsque les traitements sont lancés en mémoire vive. Dans le cas où les traitements sont lancés sur le disque dur, les temps sont «seulement» 10 fois inférieurs. La principale raison vient du fait qu'Hadoop fait sur traitement en mode lot sur le disque dur, alors que Spark peut faire du traitement en mémoire, économisant ainsi un temps considérable sur les accès disque.

- Une séquence de traitement de MapReduce ressemble à ceci : on lit des données au niveau du cluster, on exécute une opération, on écrit les résultats au niveau du cluster, on lit à nouveau les données mises à jours, on exécute l'opération suivante, etc
- Spark lit les données au niveau du cluster (ou en local), effectue toutes les opérations d'analyses nécessaires (en utilisant au maximum la RAM des machines), puis seulement écrit les résultats sur disque dur.

Pour tous les exercices, il est recommandé de consulter la documentation des RDD :

- `https://spark.apache.org/docs/latest/api/python/reference/pyspark.html#rdd-apis`
- `https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds`

Exercice I : Données textuelles (Word Count)

Q1 – Écrire un programme pyspark qui compte le nombre de lettres, de mots et de lignes dans un fichier texte.

Q2 – En terme de temps d'exécution, comparer l'approche Spark et l'approche Hadoop Map/Reduce.

Q3 – Écrire un programme pyspark qui compte la fréquence d'apparition de chaque mot dans un fichier texte.

Exercice II : Arbres remarquables

A partir du dataset sur les arbres remarquables à Paris : Cliquer ici pour le téléchargement

Q1 – Quel est le nombre d'arbres ? Votre programme doit afficher le nombres de lignes du fichier (moins le header)

Pour cela, il faut créer un RDD à partir du fichier, puis il suffit juste d'appeler la méthode `count()`. Un RDD est une sorte de collection abstraite, qui si elle est assez grande est distribuée sur plusieurs machines pour un traitement parallèle.

Il existe plusieurs façons de lancer votre script. Il faut tester tout :

- D'abord en local :
  `spark-submit programme.py` ou `spark-submit --master local[2] programme.py` .Ici on alloue deux coeurs, * permet d'utiliser tous les coeurs de la machine
- Puis sur Yarn
  `spark-submit --master yarn programme.py` Il faut s'assurer que Yarn est lancé
- Ensuite sur le Spark Master : `spark-submit --master spark://master:7077` Il faut d'abord lancer le spark master avec : `start-master.sh` puis le worker `start-worker.sh spark://idir-pc:7077` à adapter chez vous.

Q2 – Hauteur moyenne des arbres

Dans la suite de ce TP, nous allons lire des fichiers CSV, extraire et convertir des champs. C'est assez illisible de faire ces extractions en plein dans les fonctions de calcul Spark. Vous pouvez donc définir une classe utilitaire pour parser le fichier. Une classe arbre dans un fichier arbre.py (qui parse une ligne et retourne les infos voulus, par exemple getAnnee ou getHauteur, etc)

Dans votre programme pySpark, il faut importer la classe arbre. ensuite pour lancer, il faut rajouter l'option –py-files arbre.py :

- `spark-submit -master spark://master:7077 -py-files arbre.py q2.py`
- `spark-submit -py-files arbre.py q2.py`

Cela entraîne la création d'un fichier arbre.pyc contenant le code Python compilé. C'est comme un .class pour Java

Q3 – Genre du plus grand arbre

Le principe est de construire des paires (clé, valeur), avec ici la hauteur des arbres comme clé et leur genre en tant que valeur. Ensuite, on classe les paires par ordre de clé décroissante grâce à sortByKey et on garde seulement la première paire.

On voit ici une technique extrêmement importante, la manipulation de paires (clé, valeur). C'est comme avec MapReduce sur Yarn. L'algorithme repose sur le choix des clés et valeurs à associer. Il y a de nombreuses méthodes, pour information (elles ne sont pas à apprendre par coeur , juste savoir qu'on beaucoup plus d'options que Map-Reduce de Hadoop) :

- aggregateByKey, combineByKey, countByKey, foldByKey, groupByKey, reduceByKey, sortByKey, subtractByKey pour travailler avec les clés,
- countByValue, mapValues pour gérer les valeurs,
- une fonction pour extraire uniquement les clés : keys
- deux fonctions pour accéder aux valeurs : lookup et values

Refaites la même question en utilisant la fonction `max`, nettement plus performant que de faire un tri puis récupérer le premier.

Q4 – Nombre d'arbres de chaque genre

Utilisez reduceByKey puis countByKey

Exercice III : Capitales (Tiré d'un exam à Paris Saclay)

Télécharger l'archive se trouvant à cette adresse : URL. Vous y trouvez un fichier `capitals_distances.txt`. Chaque ligne du fichier est de la forme suivante :

- `PAYS1;CAPITALE1;PAYS2;CAPITALE2;DISTANCE`

Les noms de pays et de capitales sont en anglais, la distance est un nombre entier positif et représente la distance orthodromique ou "à vol d'oiseau" entre les deux capitales. De plus, le fichier est organisé de manière à ce que CAPITALE1 soit toujours une chaîne plus petite (au sens de la comparaison lexicographique des chaînes) que CAPITALE2. Par exemple, on aura une ligne :

`Germany;Berlin;France;Paris;875`

mais pas de ligne

`France;Paris;Germany;Berlin;875`

car la chaîne `"Paris"` est plus grande que `"Berlin"`.

Q1 – Calculer pour chaque ville la ville la plus proche et la ville la plus éloignée

la sortie du programme doit avoir la forme suivante $(v, vmin : d_{min}, v_{max} : d_{max})$ où v est une ville, v_{min} la ville la plus proche de v (et d_{min} la distance associée) et v_{max} la ville la plus éloignée de v (et d_{max} la distance associée).

La sortie doit être similaire (à l'ordre près) à celle donnée dans le fichier `DistMinMax.txt` se trouvant dans le même répertoire

Q2 – Écrire un job spark qui affiche pour chaque ville le nombre de villes situées à 1000km ou moins.

Attention, le calcul doit être symétrique : Paris doit être comptée dans les villes distante de moins de 1000km de Berlin, mais Berlin doit aussi être compté dans les villes distantes de moins de 1000km de Paris, bien que le fichier contienne uniquement la ligne :

```
Germany;Berlin;France;Paris;875
```

et pas la ligne

```
France;Paris;Germany;Berlin;875
```

Exercice IV : World-cities

Télécharger l'archive se trouvant à cette adresse : URL. Il contient un fichier `worldcitiespop.txt`, Regardez les premières lignes du fichier en utilisant la commande `head`. Chaque ligne du fichier contient :

- `Country,City,AccentCity,Region,Population,Latitude,Longitude`

Q1 – **Nettoyage simple :**

Écrire un programme qui nettoie le fichier `worldcitiespop.txt` pour ne conserver que les lignes ou la population n'est pas vide.

Q2 – **Statistiques :**

Écrire un programme pour afficher les statistiques suivantes sur les populations des villes. (min, max, sum, average)

Q3 – **Histogramme :**

Écrire un programme pour calculer un histogramme de fréquences des populations des villes. Pour l'histogramme on choisira les classes d'équivalences en utilisant une échelle logarithmique. (classe 0 : villes de taille [0..10[, classe 1 : villes de taille [10..100[, etc)

Q4 – **Pattern Top-K :**

Écrire un programme pour calculer et afficher les 10 villes ayant la population la plus grande

Q5 – **Re-nettoyage :**

En analysant les résultats de la question 5, on s'aperçoit que "Delhi" et "New Delhi" sont les mêmes villes, même chose pour "tokyo" et "Tokyo". Proposez une solution pour enlever les doublons (différentes villes au même endroit) en conservant les villes de plus forte population en cas de superposition. Recalculez vos histogrammes etc.