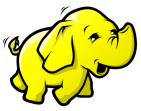
Travaux pratiques : Hadoop

Jonathan Lejeune



Objectif

Ce sujet de travaux pratiques vous permettra de vous exercer à l'utilisation de la plate-forme Hadoop. **Vous devez avoir terminé le TME préparatoire avant de commencer sujet**. Avant de commencer, assurez-vous que Hadoop est bien démarré (un nouveau formatage du HDFS est potentiellement nécessaire avant le démarrage de l'ensemble des démons).

Exercice 1 – Comprendre le découpage des inputs

Question 1

Créez les fichiers de données d'entrée suivants grâce au script *generator.sh* et le fichier *loremIpsum* qui vous ont été fournis :

```
./generator.sh loremIpsum 170
./generator.sh loremIpsum 150
./generator.sh loremIpsum 140
./generator.sh loremIpsum 20
```

Ceci permet de créer respectivement des fichiers de 170 Mo, 150 Mo, 140 Mo et 20 Mo.

Question 2

Uploader ces fichiers sur la racine du HDFS.

Question 3

Lisez et analysez le code du WordCount qui vous a été fourni et que vous avez intégré à un projet eclipse dans le TME préparatoire.

Question 4

Essayez le *WordCount* sur le fichier *loremIpsum-170* que vous avez créé dans la question 1 de cet exercice. Pendant que le job s'exécute, recontrolez les processus avec la commande *ps uxf* (pour plus de lisibilité, sauvegarder la sortie standard de cette commande dans un fichier temporaire que vous pourrez lire ensuite avec un éditeur).

Vous devez remarquer qu'il y a des processus java supplémentaires qui ont pour ancêtre commun le NodeManager qui correspondent aux containers Yarn. Combien de containers sont lancés ?

Question 5
Consultez le résultat du programme dans le dossier de sortie. Notez au passage le format des noms des fichiers de sortie.

Question 6
Identifiez le nombre de maps et le nombre de reduces que ont été lancés. Vous trouverez ces informations sur l'affichage que le job produit lorsqu'il se termine.

Question 7
Relancez un job avec les fichiers *loremIpsum-20* et comparez le nombre de maps avec le job précédent.

Question 8
Créez un dossier HDFS et placez-y les fichiers *loremIpsum-20* et *loremIpsum-150*. Comparez le nombre de maps avec les jobs précédents. Qu'en déduisez vous ?

Question 9
Quelle est la relation entre la donnée d'entrée et le nombre de maps ?

Question 10
Relancez un job avec comme entrée le fichiers *loremIpsum-140*. Le résultat est-il cohérent avec ce qu vous avez déduit à la question précédente ?

Question 11
Pour comprendre le résultat de la question 10, consultez le code source de méthode getSplits du fichier *FileInputFormat.java*. Vous pouvez trouver ce fichier en téléchargeant les sources de Hadoop sur le lien :

`http://miroir.univ-lorraine.fr/apache/hadoop/common/hadoop-3.2.1/hadoop-3.2.1-src.tar.gz`

Dans l'archive, le fichier se trouve dans le répertoire */hadoop-3.2.1-src/hadoop-mapreduce-project/ hadoop-mapreduce-client/hadoop-mapreduce-client-core/ src/main/java/org/apache/hadoop/mapreduce/lib/input/*

Question 12
A quoi sert l'optimisation détectée à la question 10 ?

Question 13
Modifiez le fichier *WordCount.java* en activant l'exécution spéculative (mettre les propriétés à true en début de programme). Refaire une nouvelle archive jar et relancez un job avec le fichier *loremIpsum-150*. Que remarquez-vous au niveau du nombre de maps ?

Exercice 2 – Écrire des programmes map-reduce

Dans cet exercice, nous allons programmer les exercice de map-reduce vu en TD. Pour tester votre code, il vous est conseillé de produire un petit fichier d'entrée de quelques lignes et de déterminer la sortie attendue et éventuellement d'utiliser le mode local de Hadoop. Si vous souhaitez le tester sur une entrée plus conséquente et/ou considérant plusieurs fichiers, vous pouvez générer des données avec le script *generateTextfile.sh* qui vous est fourni (Attention pour exécuter ce script, il est nécessaire que scala soit installé sur votre machine).

Question 1

Coder le programme Map-Reduce de la question 2 de l'exercice sur "Noodle" vu en TD.

Question 2

Coder le programme Map-Reduce du problème de Last.fm vu en TD. Pour affilier un mapper à un type de fichier particulier (utile pour le job3 qui fusionne les résultats des deux précédents qui n'ont pas le même format de sortie), il faut utiliser `MultipleInputs.addInputPath` au lieu de `FileInputFormat.addInputPath`. La classe `MultipleInputs` est une classe utilitaire qui se trouve dans le package `org.apache.hadoop.mapreduce.lib.input`. La méthode `MultipleInputs.addInputPath` prend quatre paramètres :

- `job` : le descripteur de job yarn en question
- `path` : le chemin HDFS des données d'entrée
- `inputFormatClass` : la classe du format des données d'entrée
- `mapperClass` : la classe du mapper pour ces données d'entrée

Le troisième job doit être lancé une fois que les deux précédents se sont terminés. Pensez à supprimer le fichier `_SUCCESS` présent dans chaque répertoire de sortie des 2 premier jobs avant de lancer le troisième.

Exercice 3 – Programmer dans un autre langage que java (Bonus facultatif)

Nous allons désormais utiliser la fonctionnalité de streaming de Hadoop pour pouvoir exécuter des programmes Map-Reduce sans l'API Java de base.

Question 1

Pour utiliser la fonctionnalité du streaming, nous allons utiliser une archive jar fourni dans les librairie de la plate-forme. Le chemin de cette archive est :

```
$MY_HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar
```

Prenez connaissance des options de cette commandes en tapant

```
yarn jar  $MY_HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar  streamjob -help
```

Nous avons vu en cours que les communications entre la JVM container et le processus streaming se faisait par les entrées et les sorties standards de celui-ci.

Analysons maintenant la valeur des différents couples <clé, valeur> transmis aux entrées des maps et des reduce.

Question 2

Nous allons dans un premier temps nous intéresser aux informations transmises aux maps dans le cas d'un fichier d'entrée au format binaire. Créez un fichier binaire sur le HDFS grâce au programme java *SequenceFileCreator* : le compiler, l'archiver en jar et le lancer avec la commande yarn :

```
yarn jar <votre archive jar> SequenceFileCreator /in 50
```

créera un fichier le fichier binaire */in* et contiendra 50 entrées au format binaire. Vous pourrez ensuite constatez que le contenu de ce fichier (*hdfs dfs -cat /in*) n'est pas lisible par le terminal.

Question 3

Que fait la commande suivante :

```
yarn jar  $MY_HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar  streamjob \
       -input /in \
       -output /out \
       -mapper '/bin/sed -e  "s/^\(.*\)$/LIGNE_LUE:\1/"' \
       -numReduceTasks 0 \
       -inputformat SequenceFileAsTextInputFormat
```

Question 4

Lancez cette commande puis constatez ensuite le contenu du ou des fichiers de sortie. En déduire le format des couples clés/valeurs transmis aux maps ?

Question 5

Refaire la même chose mais avec un fichier texte tel que le fichier *loremIpsum* et en remplaçant

```
-inputformat SequenceFileAsTextInputFormat
```

par

```
-inputformat TextInputFormat
```

Ne pas oublier le cas échéant de changer le nom de l'input. Que constatez-vous ?

Question 6

Nous allons dans un premier temps nous intéresser aux informations transmises aux reduces. Reprendre le fichier binaire créé à la question 2 et lancer la commande :

```
yarn jar  $MY_HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar  streamjob \
       -input /in \
       -output /out \
       -mapper '/bin/cat' \
       -reducer '/bin/sed -e  "s/^\(.*\)$/LIGNE_LUE:\1/"' \
       -numReduceTasks 1 \
       -inputformat SequenceFileAsTextInputFormat
```

Affichez le contenu du fichier de sortie et en déduire comment sont transmises les informations <clé,list<valeur» au reduce.