TP Hadoop Python (4h)

L. Benyoussef & S. Derrode

1. Prise en main

Chaque élève dispose d'un compte pour accéder au serveur Big-Data/Hadoop de la plateforme Amigo de l'École Centrale de Lyon :

mso31_y, où *y* désigne le numéro qui vous sera attribué par l'encadrant du TP.

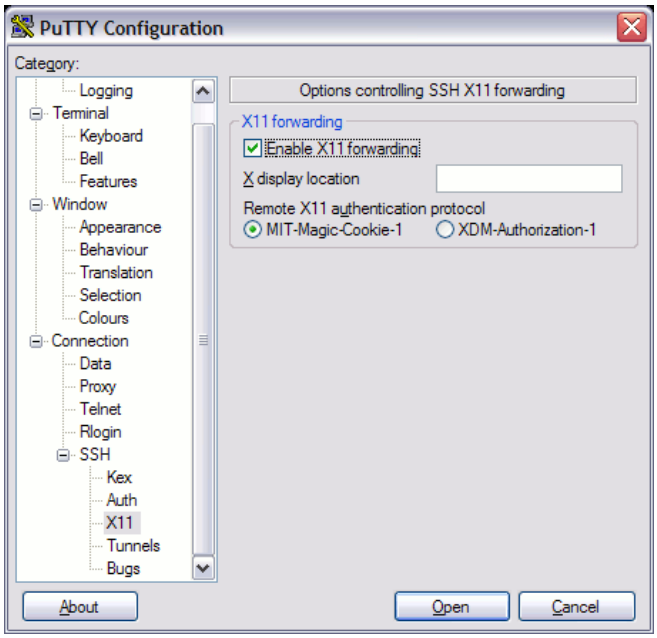
Le mot de passe est identique au compte. Ce compte vous est accessible durant toute la durée du module. Il sera ensuite détruit dans le courant du mois d'avril. Je vous enverrai un mail pour faire une éventuelle copie de votre compte.

Pour vous connecter à votre compte, veuillez suivre la procédure suivante. C'est la même que celle présentée lors du MOD, aussi cette annexe est-elle essentiellement disponible pour les élèves n'ayant pas suivi le MOD.

Pour vous connecter sur votre compte, vous devez auparavant avoir installé les logiciels suivants :

Pour Windows :

- **Remarque** : il semble que Windows 10 a un client est déjà installé de base. Il suffit de lancer les commande ssh comme avec linux ou mac, cf. ci-dessous.
- Sinon, installez *putty* (http://www.putty.org/) et lance-le. Dans le Menu SSH/X11, cochez la case X11 comme ceci :



Dans le menu *Session*, entrez l'adresse de l'hôte : 156.18.90.100, puis *open*. Loggez-vous avec votre loggin et votre password.

Pour Mac/Linux:

- Dans un terminal, tapez « *ssh -Y loggin@156.18.90.100* » où *loggin* est votre compte. Entrez ensuite le mot de passe.

Vous devriez alors voir apparaître un message similaire à :

```
>> ssh -Y sderrode@156.18.90.100
sderrode@156.18.90.100's password:
Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Sun Jan  8 07:25:06 2017 from vpn96-192.vpnusers.ec-lyon.fr
Welcome to Bright release         7.3
                                                       Based on CentOS Linux 7
                                                                  ID: #000002
Use the following commands to adjust your environment:
'module avail'            - show available modules
'module add <module>'     - adds a module to your environment for this session
'module initadd <module>' - configure module to be loaded at every login
-------------------------------------------------------------------------------
[sderrode@bright73 ~]$
```

On peut apprendre, grâce aux traces, que le serveur fonctionne sous *CentOS Linux 7* (une distribution de *Linux* parmi d'autres), et que le système de gestion du cluster *Hadoop* s'appelle *Bright* (ou *Bright Cluster Manager*).
Ça y est, vous êtes connecté au serveur de la DSI qui simule un data center. Comme vous pouvez le constater, il n'y a pas d'interface graphique, et la manipulation des fichiers doit se faire à la main. Il faut donc connaître un minimum de commandes *Linux* à taper sur le Terminal (aussi appelé 'invite de commande' ou 'console' en fonction des systèmes d'exploitation).

Avant de commencer, vous devez télécharger l'ensemble des fichiers (de tous les TP), grâce à un dépôt *github*. Pour se faire, ouvrez un terminal et déplacer vous dans un répertoire de travail. Tapez ensuite les commandes suivantes :

git clone https://github.com/SDerrode/tp-hadoop-python.git
cd tp-hadoop-python
ls

Vous verrez apparaître plusieurs répertoires. Entrez dans le répertoire « *Hadoop* » qui correspond au répertoire du TP en cours.

2. WordCount, Hadoop Streaming

Note : Pour éviter les problèmes de copier-coller entre l'énoncé (pdf) et votre terminal, j'ai concaténé toutes les commandes dans un fichier « Commandes.txt » que vous trouverez dans le répertoire en cours.

Il s'agit ici d'exécuter les programmes vus en cours, dans différentes configurations, sur les 5 tomes de l'œuvre de V. Hugo, « Les Misérables ».

1. Exécution en local,
   *chmod +x *.py* (rendre exécutable tous les scripts python)
   cat ../LesMiserables/LesMiserables_T4.txt | wc_mapper.py
   cat ../LesMiserables LesMiserables_T4.txt | wc_mapper.py | sort -k1,1 | wc_reducer.py
   cat ../LesMiserables LesMis.txt | wc_mapper.py | sort -k1,1 | wc_reducer.py*
2. Copiez les 5 tomes sur votre espace HDFS, en utilisant les commandes suivantes :
   hdfs dfs -mkdir LesMiserables
   hdfs dfs -put LesMis.txt LesMiserables*
3. Exécution sur HDFS
   export STREAMINGJAR='/cm/shared/apps/hadoop/Hortonworks/2.7.3.2.5.0.0-1245/share/hadoop/tools/lib/hadoop-streaming-2.7.3.2.5.0.0-1245.jar'
   hadoop jar $STREAMINGJAR -files wc_mapper.py,wc_reducer.py -mapper wc_mapper.py -reducer wc_reducer.py -input LesMiserables/LesMis.txt -output sortie*
   hdfs dfs -ls sortie (vous devriez voir un fichier *_SUCCESS*, vide mais dont la seule présence marque le succès du job, et un autre appelé *part-0000*, contenant les résultats du job).
   hdfs dfs -cat sortie/part-00000
   hdfs dfs -rm -r -f sortie (pour supprimer le répertoire de sortie entre 2 exécutions)
4. Avec plusieurs reducers

hadoop jar $STREAMINGJAR -D mapred.reduce.tasks=3 -input LesMiserables/LesMis.txt -output sortie -mapper /bin/cat -reducer /bin/wc*
hdfs dfs -ls sortie
hdfs dfs -cat sortie/part-00001
hdfs dfs -rm -r -f sortie

5. Avec un combiner
   hadoop jar $STREAMINGJAR -files wc_mapper.py,wc_reducer.py -mapper wc_mapper.py -combiner wc_reducer.py -reducer wc_reducer.py -input LesMiserables/LesMis.txt -output sortie*
   Vérifiez, en comparant les traces, que les volumes de données transférés entre mappers et reducers sont moindre que ceux engendrés par la version sans combiner !
6. Tester les programmes wc_mapper_improved.py et wc_reducer_improved.py
   hadoop jar $STREAMINGJAR -files wc_mapper_improved.py,wc_reducer_improved.py -mapper wc_mapper_improved.py -reducer wc_reducer_improved.py -combiner wc_reducer_improved.py -input LesMiserables/LesMiserables_T1.txt -output sortie
7. Modifier le programme précédent de telle manière que la sortie ne contienne que les mots dont la fréquence est supérieure à une valeur fixée (par exemple 10). N'oubliez pas de rendre exécutable vos nouveaux fichiers python (« *chmod +x zzz.py* »).

3. LIBRAIRIE MRJOB

Documentation : https://media.readthedocs.org/pdf/mrjob/latest/mrjob.pdf

1. Tester les programmes *wc1_mrjob.py* et *wc2_mrjob.py*, en utilisant les commandes suivantes :
   python3 wc1_mrjob.py ../LesMiserables/LesMiserables_T1.txt
   python3 wc1_mrjob.py ../LesMiserables/LesMiserables_T4.txt --mapper
   python3 muw_mrjob.py ../LesMiserables/LesMiserables_T1.txt -r local
   python3 wc2_mrjob.py ../LesMiserables/LesMiserables_T4.txt -r hadoop
   python3 muw_mrjob.py hdfs:///user/sderrode/LesMiserables/LesMiserables_T1.txt -r hadoop (n'oubliez pas d'adapter le *loggin* à votre compte)
   python3 muw_mrjob.py hdfs:///user/sderrode/LesMiserables/LesMiserables_T1.txt -r hadoop --output-dir hdfs:///user/sderrode/sortie/ --no-output
   hdfs dfs -ls sortie
   hdfs dfs -cat sortie/part-00000
   python3 wc2_mrjob.py LesMiserables_T4.txt -r local --jobconf mapred.red.tasks=5 --output-dir sortie --no-output
   hdfs dfs -ls sortie
   hdfs dfs -text sortie/part-0002
2. Il s'agit maintenant de développer un algorithme hadoop/mrjob permettant de multiplier une matrice (supposée très grande) avec un vecteur de dimension adéquate (pas de contrôle sur les dimensions). Nous appellerons la classe *MRMatrice*.
   Voici les différentes étapes pour arriver au but :
   a. La matrice est codée dans un fichier appelé matrice.txt de la manière suivante :
      0 1 2 3
      1 2 3 4
      2 3 4 5
      La première colonne représente le numéro de ligne, alors que les 3 autres colonnes représentent les valeurs de la matrice. Puisqu'elles sont numérotées, l'ordre des lignes n'a ici que peu d'importance.
   b. Outre les méthodes *mapper*, *combiner* et *reducer* que nous avons déjà vues, MRJob prévoit une méthode dont la signature est « *def mapper_init(self)* ». Cette méthode est appelée une unique fois au début de chaque *mapping*. Utilisez cette méthode pour définir le vecteur à multiplier :

i. Dans une première version de l'algorithme, le vecteur sera codé en dur, sous forme d'une liste de valeurs, par exemple « vecteur = [2, 4.5, 6] », pour faire vos tests. Le programme python sera appelé *matrice1_mrjob.py.*

ii. Dans une seconde version, *matrice2_mrjob.py*, le vecteur sera lu dans un fichier dont le nom sera passé sous forme un argument lors de l'exécution :

python3 matrice2_mrjob.py matrice.txt --vector vector.txt -r inline

Pour cela, vous pouvez utiliser le système prévu par MRJob pour transférer des options au programme, en ajoutant la méthode suivante :

```
def configure_args(self):
        super(MRMatrice, self).configure_args()
        self.add_file_arg('--vector')
```

sachant qu'il est possible de retrouver le contenu transféré au programme par le biais de l'option « *vector* » (c'est à dire le nom du fichier) grâce à l'instruction « *self.options.vector* ».

c. Intégrez à votre programme un *combiner* pour réduire les transferts de données entre mappers et reducers. Le programme Python correspondant sera appelé *matrice3_mrjob.py. Remarque* : Si le *combiner* est identique au *reducer*, il suffit d'écrire « combiner = reducer » dans la classe. Il n'est donc pas besoin de recopier la méthode !

d. Tester le programme sur la GROSSE matrice et le GROS vecteur stockés dans le chemin suivant : *hdfs:///user/shared/* (une copie de ces 2 fichiers se trouvent dans le répertoire courant). Constatez que les données du vecteur de sortie ne sont pas nécessairement dans le bon ordre. Y a-t-il un moyen de régler ce problème (cf. méthode « *reducer_final* » dans https://media.readthedocs.org/pdf/mrjob/latest/mrjob.pdf)?

4. PROJET : REDUCE SIDE JOIN

(depuis le tutorial : https://www.edureka.co/blog/mapreduce-example-reduce-side-join/)

Nous allons ici développer un programme MapReduce pour réaliser une opération de jointure (ala SQL). Une opération de jointure consiste à combiner 2 bases de données. Par exemple, les entreprises maintiennent généralement deux tables séparées pour leurs clients et pour les achats réalisés par ces clients. Il est alors bien souvent nécessaire de fusionner ces 2 tables pour calculer des quantités comme « Quelle la plus grosses somme dépensée par un client ce mois-ci ». Pour répondre à cette question, il faut réaliser une jointure entre ces deux tables séparées en utilisant une colonne commune (la clé étrangère), comme l'identifiant du client.

Les deux fichiers associés à cet exercice se nomment : « *cust_details* » et « *transaction_details* », et présente les résultats de gestion d'un réseau de salles de fitness. Nous chercherons à connaître :

- La fréquence de visite des salles de sport de chaque client (quelle que soit la salle) ;
- les sommes dépensées par chaque client depuis que le fichier de transactions existe.

Dans le « *reduce side join* », le *reducer* est responsable de la jointure. Il est plus facile à implémenter que le « *map side join* » car les phases de mapping et de shuffling envoient des paires qui ont la même clé au même *reducer*, et les données sont ainsi organisées par défaut.
Truc : vous pouvez implémenter deux *mappers* : l'un pour le fichier « *cust-details* », l'autre pour le fichier « *transaction_details* ».

Notes finales :

- La contrepartie de la simplicité du « *reduce side join* », c'est qu'il génère un énorme trafic réseau.
- Quand on veut réaliser une telle jointure, on préfèrera utiliser Apache Hive (écosystème Hadoop), qui est un langage SQL-like destiné à traiter de gros volumes de données HDFS.
- Il existe une version « *map side join* » de la jointure, plus performant lorsque l'un des deux fichiers est beaucoup plus petit que le second, et qu'il peut tenir en mémoire vive.