TP Hadoop : MapReduce en Python

1. Préparation des données sur HDFS

1. Télécharger le fichier purchases.txt

Créez un répertoire dans HDFS, appelé `myinput`. Pour cela, tapez :

```
hadoop fs -mkdir myinput
```

Pour copier le fichier `purchases.txt` dans HDFS sous le répertoire `myinput`, il s'agit e se placer dans le répertoire local `data` où se trouve le fichier, puis tapez la commande :

```
hadoop fs -put purchases.txt myinput/
```

Pour afficher le contenu du répertoire `myinput`, la commande est :

```
hadoop fs -ls myinput
```

Pour **visualiser les dernières lignes du fichier**, tapez :

```
hadoop fs -tail myinput/purchase.txt
```

`hadoop fs -ls`	Afficher le contenu du répertoire racine
`hadoop fs -put file.txt`	Upload un fichier dans hadoop (à partir du répertoire courant linux)
`hadoop fs -get file.txt`	Download un fichier à partir de hadoop sur votre disque local
`hadoop fs -tail file.txt`	Lire les dernières lignes du fichier
`hadoop fs -cat file.txt`	Affiche tout le contenu du fichier
`hadoop fs -cat file.txt | less`	Lire le fichier page par page
`hadoop fs -mv file.txt newfile.txt`	Renommer le fichier
`hadoop fs -rm newfile.txt`	Supprimer le fichier
`hadoop fs -mkdir myinput`	Créer un répertoire
`hadoop fs -rm -f -r myinput`	Supprime un répertoire, et son contenu récursivement

TABLE 1 – Principales commandes de manipulation de fichiers HDFS

1 Map Reduce

Map Reduce est un patron d'architecture de développement permettant de traiter les données volumineuses de manière parallèle et distribuée.
Il se compose principalement de deux types de programmes :

- Les Mappers permettent d'extraire les données nécessaires sous forme de clef/valeur, pour pouvoirensuite les trier selon la clef.
- Les Reducers prennent un ensemble de données triées selon leur clef, et effectuent le traitement nécessaire sur ces données (somme, moyenne, total...).

4.1 Mapper

Soit un fichier comportant 6 champs, séparés par des tabulations. Le Mapper doit :

- Séparer les différents champs par tabulation
- Extraire les éléments voulus à partir de ces champs, sous forme de clef/valeur

Pour ce premier exercice, notre but est de déterminer le total des ventes par magasin, pour un fichier log dont les champs sont de la forme suivante :
`date > temps > magasin > produit > co t > paiement`
Pour calculer les ventes par magasin, le couple (clef, valeur) à extraire est `(magasin,co t)`.
Pour faire cela, le code du Mapper est le suivant :

```
#!/usr/bin/python

# Format of each line is:
# date\ttime\tstore name\titem description\tcost\tmethod of payment
#
# We want elements 2 (store name) and 4 (cost)
# We need to write them out to standard output, separated by a tab

import sys
```

```
for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print "{0}\t{1}".format(store, cost)
```

4.2 Reducer

Le Reducer permet de faire le traitement désiré sur des entrées sous forme de clef/valeur, préalablement triées par Hadoop (on n'a pas à s'occuper du tri manuellement). Dans l'exemple précédent, une fois que le Mapper extrait les couples (store,cost), le Reducer aura comme tâche de faire la somme de tous les coûts pour un même magasin. Le code du Reducer est le suivant :

```
#!/usr/bin/python

# Format of each line is:
# date\ttime\tstore name\titem description\tcost\tmethod of payment
#
# We want elements 2 (store name) and 4 (cost)
# We need to write them out to standard output, separated by a tab

import sys

salesTotal = 0
oldKey = None

# Loop around the data
# It will be in the format key\tval
# Where key is the store name, val is the sale amount
```

```
#
# All the sales for a particular store will be presented,
# then the key will change and we'll be dealing with the next store

for line in sys.stdin:
    data_mapped = line.strip().split("\t")
    if len(data_mapped) != 2:
        # Something has gone wrong. Skip this line.
        continue

    thisKey, thisSale = data_mapped

    if oldKey and oldKey != thisKey:
        print oldKey, "\t", salesTotal
        oldKey = thisKey;
        salesTotal = 0

    oldKey = thisKey
    salesTotal += float(thisSale)

if oldKey != None:
    print oldKey, "\t", salesTotal
```

4.3 Lancer un Job entier

Lancer un job entier sur Hadoop implique qu'on fera appel au mapper puis au reducer sur une entrée volumineuse, et qu'on obtiendra à la fin un résultat, directement sur HDFS. Pour faire cela, l'instruction à exécuter est :

```
hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-
2.0.0-mr1-cdh4.1.1.jar -mapper mapper.py -reducer reducer.py -file mapper.py
-file reducer.py -input myinput -output joboutput
```

Cette instruction donne en paramètres les fichiers correspondant aux Mappers et Reducers, et les répertoires contenant le fichier d'entrée (`myinput`) et la sortie à générer (`joboutput`). Le répertoire de sortie, après exécution, contiendra un fichier appelé `part-00000`, représentant la sortie désirée.

Remarque 1 : Nous utilisons `Hadoop Streaming` qui permet de créer et lancer des jobs MapReduce avec tout type d'exécutable ou script en tant que mapper et reducer. La manière standard est d'écrire des programmes MapReduce en Java via l'API Java MapReduce. Ici nos scripts sont écrits en Python, mais les mappers et reducers pourraient être des classes Java, des utilitaires unix, des scripts R, Ruby, etc. Les Mappers liront les données fournies dans le flux standard d'entrée unix `stdin` et les réécriront dans la sortie standard `stdout` via `print`.