LabWork 2

Implémentation de MapReduce

1. Objectifs

Nous allons écrire un programme MapReduce simple pour Hadoop en Python, mais sans utiliser Jython pour traduire notre code en fichiers jar Java.

Notre programme imitera le WordCount, c'est-à-dire qu'il lira des fichiers texte et comptera la fréquence d'apparition des mots. L'entrée consiste en des fichiers texte, et la sortie sera également des fichiers texte, où chaque ligne contient un mot et le nombre de fois où il est apparu, séparés par une tabulation.

2. Code MapReduce

Le but derrière le code Python suivant est que nous allons utiliser l'API Hadoop Streaming pour nous aider à transmettre des données entre nos codes Map et Reduce via STDIN (entrée standard) et STDOUT (sortie standard). Nous utiliserons simplement sys.stdin de Python pour lire les données d'entrée et redirigeons notre propre sortie sur sys.stdout.

2.1. Fonction Map

Enregistrez le code suivant dans le fichier « mapper.py » (sur votre répertoire de base). Il lira les données de STDIN, les divisera en mots et émettra une liste de lignes associant les mots à leur nombre (intermédiaire) sur STDOUT. Le script Map ne calculera pas la somme (intermédiaire) des occurrences d'un mot. Au lieu de cela, il produira des tuples <mot> 1 - même si un mot spécifique peut apparaître plusieurs fois dans l'entrée. Dans notre cas, nous laissons l'étape suivante de réduction faire la somme finale.

Assurez-vous que le fichier a la permission d'exécution (chmod +x mapper.py devrait faire l'affaire).

```
#!/usr/bin/python3
"""mapper.py"""

import sys

# Entrée vient de l'entrée standard
for line in sys.stdin:
```

```
    # Suppression des espaces
    line = line.strip()
    # Diviser les lignes en mots
    words = line.split()
    # Créer les couples
    for word in words:
        print '%s\t%s' % (word, 1)
```

2.2. Fonction Reduce

Enregistrez le code suivant dans le fichier «reducer.py». Il lira les résultats de mapper.py à partir de STDIN (le format de sortie de mapper.py et le format d'entrée attendu de reducer.py doivent donc correspondre) et additionnera les occurrences de chaque mot pour obtenir un nombre final, puis affichera ses résultats sur STDOUT.

Assurez-vous que le fichier a la permission d'exécution (chmod +x reducer.py devrait faire l'affaire) ou vous rencontrerez des problèmes.

```
#!/usr/bin/python3
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# Entrée vient de l'entrée standard
for line in sys.stdin:
    # supprimer les espaces de début et de fin
    line = line.strip()

    # découper les lignes reçues du mapper.py en mot et occurnece
    word, count = line.split('\t', 1)

    # Convertir la variable count en entier
    try:
        count = int(count)
    except ValueError:
        # En cas de problème de conversion
        # ignorer la ligne
```

```
        continue

    # Ce commutateur IF fonctionne uniquement parce que Hadoop trie la sortie
    # de la phase Map par clé (ici : le mot) avant de la transmettre au
    # réducteur (reducer)
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # Afficher le résultat sur STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# n'oubliez pas d'afficher le dernier mot si nécessaire
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

2.3. Tester le code (cat data | map | sort | reduce)

On recommande de tester les scripts « mapper.py » et « reducer.py » localement avant de les utiliser dans un Job MapReduce.

NB. Il faut faire attention aux chemins des scripts « mapper.py » et « reducer.py »

Lancez les commandes suivantes sur votre terminal :

```
echo "FPS Safi FP Safi FPS SDA" | ./mapper.py
echo "FPS Safi FP Safi FPS SDA" | ./mapper.py | sort -k1,1 | ./reducer.py
cat fichier.txt | ./mapper.py
```

3. Lancer le code sur Hadoop

3.1. Préparation des données

Nous utiliserons trois livres électroniques du Projet Gutenberg pour cet exemple à savoir :

Livre	Lien
The Outline of Science, Vol. 1 (of 4) by J. Arthur Thomson	http://www.gutenberg.org/etext/20417
The Notebooks of Leonardo Da Vinci	http://www.gutenberg.org/etext/5000
Ulysses by James Joyce	http://www.gutenberg.org/etext/4300

Avant d'exécuter le Job MapReduce proprement dit, nous devons d'abord copier les fichiers de notre système de fichiers local vers le HDFS d'Hadoop.

Syntaxe

```
hdfs dfs -copyFromLocal localFile hdfsFile
```

Exemple

```
hdfs dfs -copyFromLocal ~/20417.txt /data/file1.txt
```

NB. Il faut créer le répertoire /data en premier sur le cluster HDFS, la commande à utiliser est :

```
hdfs dfs -mkdir /data/file1.txt
```

3.2. Lancer le Job MapReduce

Maintenant que tout est prêt, on peut exécuter le job MapReduce Python sur le cluster Hadoop. Comme susmentionné, on utilise l'API Hadoop Streaming pour nous aider à transmettre les données entre les codes Map et Reduce via STDIN et STDOUT.

```
hadoop jar hadoop-*streaming*.jar
    -mapper mapper.py
    -reducer reducer.py
   -input /data/gutenberg/*
   -output /result
```

Le job lira tous les fichiers dans le répertoire HDFS /data, les traitera et stockera les résultats dans le répertoire HDFS /result. En général, Hadoop crée un fichier de sortie par réducteur ; dans notre cas, il ne crée qu'un seul fichier car les fichiers d'entrée sont très petits.

Quel est le mot le plus apparent dans ces livres ?

4. A faire : Traitement de données météorologiques

Vous recevez un fichier contenant des données météorologiques sur les villes de toute la planète. Vous devez implémenter deux jobs de map-reduce qui :

- Calculer la température maximale pour chaque année
- Calculer le nombre de mois qui ont eu une température supérieure à une valeur donnée (un paramètre).
- Le format d'une ligne dans le fichier est : jour : mois : année : température : ville