Rapport de TP : Traitement de Données avec MapReduce et Hadoop

Par Jean Marie Fande NDIAYE

Enseignant : Dr. Djibril Mboup

Cours : BigData – ING3 GEIT IPSL 2024

Table des matières

1. Introduction et Objectifs

Ce TP vise à exploiter un cluster Hadoop pour effectuer des traitements MapReduce. MapReduce, développé par Google, est un modèle de programmation permettant le traitement parallèle de grandes quantités de données. Nous utiliserons Hadoop, qui fournit une implémentation de MapReduce, pour exécuter un job de comptage des occurrences de mots dans un document, à l'aide du programme `WordCount` en Python.

2. Fonctionnement de MapReduce : Introduction au Programme WordCount

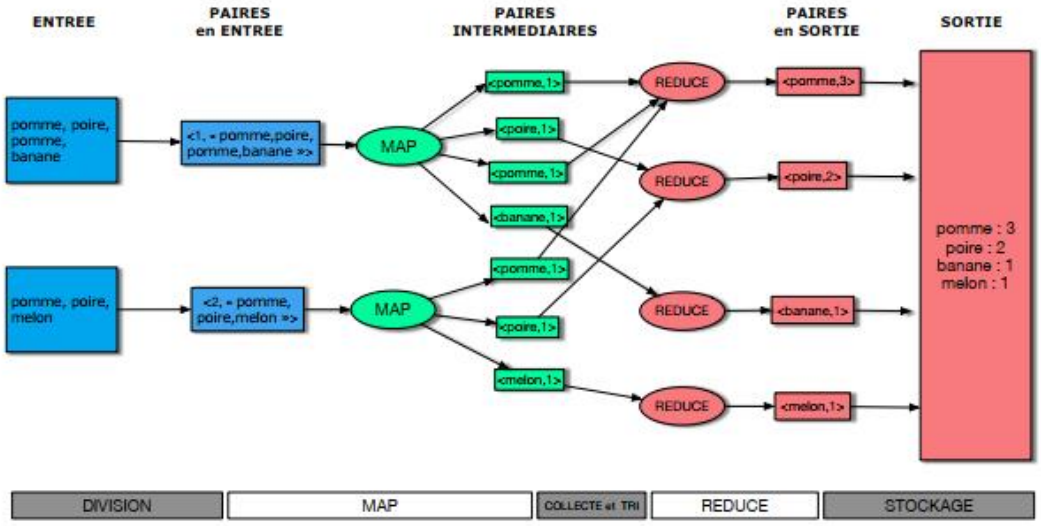
MapReduce est un modèle de programmation qui facilite le traitement parallèle de grandes quantités de données. À l'instar du programme "Hello World" dans les langages de programmation, MapReduce dispose de son propre exemple introductif : le programme `WordCount`. Ce programme sert de première approche pour comprendre la syntaxe et les principes fondamentaux du modèle MapReduce. L'objectif principal de `WordCount` est de compter le nombre d'occurrences des mots dans un document ou une collection de documents.

Le programme `WordCount` illustre les étapes du traitement MapReduce de manière intuitive :

1. **Étape Map :** La fonction `Map` prend en entrée des chaînes de caractères et les découpe en mots individuels. Pour chaque mot, dénoté par `w`, elle émet une paire de la forme `(w, 1)`. Ici, `w` est le mot trouvé et `1` représente une occurrence de ce mot.
2. **Étape Reduce :** Les paires générées par l'étape `Map` sont ensuite regroupées par clé, ce qui signifie que toutes les occurrences du même mot sont rassemblées. La fonction `Reduce` reçoit alors ces paires regroupées et calcule le total des occurrences pour chaque mot. Pour chaque mot, la fonction `Reduce` renvoie une paire où la clé est le mot et la valeur est le nombre total d'occurrences dans le document ou la collection de documents.

Ce processus est illustré dans la Figure 1, qui détaille les différentes étapes du traitement MapReduce à l'aide de la tâche `WordCount`. En résumé, `WordCount` démontre comment MapReduce permet de traiter et d'analyser de grandes quantités de données en parallèle, en simplifiant les tâches complexes en étapes de transformation et d'agrégation des données.

Figure 1 : Les étapes d'un processus MapReduce à l'aide de la tâche WordCount.



3. Installation et Configuration

3.1 Installation du Cluster Hadoop

Pour mettre en place un environnement Hadoop fonctionnel et exécuter des traitements MapReduce, il est nécessaire d'installer et de configurer un cluster Hadoop. Voici les étapes à suivre pour réaliser cette installation :

1. **Préparation de l'Environnement**

Avant d'installer Hadoop, vous devez préparer l'environnement virtuel dans lequel Hadoop sera exécuté. Utilisez **VirtualBox** et **Vagrant** pour créer une machine virtuelle (VM) qui servira de cluster Hadoop.

VirtualBox : Téléchargez et installez VirtualBox depuis VirtualBox. VirtualBox vous permettra de créer et de gérer des machines virtuelles sur votre machine physique.

Vagrant : Téléchargez et installez Vagrant depuis Vagrant. Vagrant facilitera la gestion de la configuration et le déploiement de la machine virtuelle.

2. Clonage du Dépôt

Clonez le dépôt contenant le fichier de configuration nécessaire pour déployer le cluster Hadoop. Assurez-vous d'avoir `git` installé sur votre machine. Utilisez la commande suivante pour cloner le dépôt :

```
git clone https://github.com/sopeKhadim/hadoopVagrant.git
```

3. Lancement de la Machine Virtuelle

Déplacez-vous dans le répertoire où se trouve le fichier `Vagrantfile` :

```
cd <chemin_du_répertoire_avec_Vagrantfile>
```

Lancez la machine virtuelle en utilisant Vagrant :

```
vagrant up
```

Cette commande téléchargera l'image de la machine virtuelle, la configurera et la démarrera. La machine virtuelle sera configurée avec un environnement prêt à l'emploi pour Hadoop.

4. Connexion à la Machine Virtuelle

Une fois que la machine virtuelle est en cours d'exécution, connectez-vous à celle-ci en utilisant la commande suivante :

```
vagrant ssh
```

Cette commande ouvrira une session SSH dans la machine virtuelle, vous permettant d'accéder à l'environnement.

3.2 Installation des Bibliothèques

1. **Installation de `mrjob` :**

mrjob est une bibliothèque Python conçue pour simplifier le développement et l'exécution de jobs MapReduce. Développée par Yelp, cette bibliothèque nous permet de créer des programmes MapReduce en Python, facilitant ainsi le traitement de données à grande échelle tout en tirant parti des fonctionnalités offertes par le langage Python.

```
easy_install mrjob
```

4. Écriture et Exécution du Job MapReduce

4.1 Code source

Le script Python `WordCount.py` est le suivant :

```python
# -*- coding: utf-8 -*-

from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w']+")

class WordCount1(MRJob):

    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_words,
                   reducer=self.reducer_count_words)
        ]

    def mapper_get_words(self, _, line):
        # Découpeage de la chaîne et itération sur chaque mot.
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def reducer_count_words(self, word, counts):
        # On définit une variable intermédiaire pour stocker le nombre
d'occurrences
        yield (word, sum(counts))

if __name__ == '__main__':
    WordCount1.run()
```

4.2 Préparation des Données

1. **Lancement du Cluster Hadoop :**

   ```
   start-dfs.sh
   start-yarn.sh
   ```

2. **Téléchargement et Préparation des Données :**

Télécharger le fichier `ancien_figaro.txt` et le placer dans `/home/vagrant/shareFolder/`.

3. **Copie des Données dans HDFS :**

```
hdfs dfs -mkdir -p /user/hadoop/
```

```
hdfs dfs -put /home/vagrant/shareFolder/ancien_figaro.txt
/user/hadoop/
```

```
[vagrant@localhost home]$ hdfs dfs -put /home/vagrant/shareFolder/ancien_figaro.txt /user/hadoop/
put: '/user/hadoop/': No such file or directory: 'hdfs://localhost:9000/user/hadoop'
[vagrant@localhost home]$ hdfs dfs -mkdir -p /user/hadoop/
[vagrant@localhost home]$ hdfs dfs -put /home/vagrant/shareFolder/ancien_figaro.txt /user/hadoop/
2024-07-19 00:26:49,821 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
[vagrant@localhost home]$ hdfs dfs -ls /user/hadoop/
Found 1 items
-rw-r--r--   1 vagrant supergroup     522055 2024-07-19 00:26 /user/hadoop/ancien_figaro.txt
```

4.3 Exécution du Job MapReduce

1. **Exécution du Job avec `mrjob` :** Soumettre le job à Hadoop :

```
python WordCount.py -r hadoop hdfs:///user/hadoop/ancien_figaro.txt -
o hdfs:///user/hadoop/output
```

sortie:

```
[vagrant@localhost shareFolder]$ python WordCount.py -r hadoop hdfs:///user/
hadoop/ancien_figaro.txt -o hdfs:///user/hadoop/output
No configs found; falling back on auto-configuration
No configs specified for hadoop runner
Looking for hadoop binary in /opt/hadoop/bin...
Found hadoop binary: /opt/hadoop/bin/hadoop
Using Hadoop version 3.2.1
Looking for Hadoop streaming jar in /opt/hadoop...
Found Hadoop streaming jar: /opt/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.2.1.jar
Creating temp directory /tmp/WordCount.vagrant.20240719.011203.407301
uploading working dir files to hdfs:///user/vagrant/tmp/mrjob/WordCount.vagrant.20240719.011203.407301/files/wd...
Copying other local files to hdfs:///user/vagrant/tmp/mrjob/WordCount.vagrant.20240719.011203.407301/files/
Running step 1 of 1...
  packageJobJar: [/tmp/hadoop-unjar6715198535264967943/] [] /tmp/streamjob8660867222631578809.jar tmpDir=null
  Connecting to ResourceManager at /0.0.0.0:8032
  Connecting to ResourceManager at /0.0.0.0:8032
  Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/vagrant/.staging/job_1721348138477_0001
  SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
  Total input files to process : 1
  SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
  SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
  number of splits:2
  SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
  Submitting tokens for job: job_1721348138477_0001
  Executing with tokens: []
  resource-types.xml not found
  Unable to find 'resource-types.xml'.
  Submitted application application_1721348138477_0001
  The url to track the job: http://localhost:8088/proxy/application_1721348138477_0001/
  Running job: job_1721348138477_0001
  Job job_1721348138477_0001 running in uber mode : false
   map 0% reduce 0%
   map 50% reduce 0%
   map 100% reduce 0%
   map 100% reduce 100%
  Job job_1721348138477_0001 completed successfully
  Output directory: hdfs:///user/hadoop/output
Counters: 55
        File Input Format Counters
```

5. Vérification des Résultats

1. **Vérification du Répertoire de Sortie sur HDFS :**

```
hdfs dfs -ls /user/hadoop/output/
```

```
[vagrant@localhost shareFolder]$ hdfs dfs -ls /user/hadoop/output/
Found 2 items
-rw-r--r--   1 vagrant supergroup          0 2024-07-19 01:14 /user/hadoop/output/_SUCCESS
-rw-r--r--   1 vagrant supergroup     145693 2024-07-19 01:14 /user/hadoop/output/part-00000
[vagrant@localhost shareFolder]$
```

2. **Affichage du Contenu des Résultats :**

```
hdfs dfs -head /user/hadoop/output/part-00000
```

```
[vagrant@localhost shareFolder]$ hdfs dfs -head /user/hadoop/output/part-00000
2024-07-19 01:18:21,697 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
"'as"   1
"000"   23
"1"     53
"10"    11
"100"   1
"1001"  1
"11"    6
"12"    12
"13"    4
"134"   1
"139"   1
"14"    10
"15"    8
"1500"  1
"1559"  1
"16"    7
"1631"  1
"1650"  1
"1672"  1
"17"    4
"172"   1
"1789"  1
"18"    7
"1814"  1
"1815"  8
"1816"  1
"1823"  1
"1824"  3
"1826"  23
"1827"  32
"1828"  16
"1829"  23
"1830"  12
"1832"  1
"1834"  1
"1846"  1
"1848"  1
"1850"  1
```

```
hdfs dfs -tail /user/hadoop/output/part-00000
```

```
[vagrant@localhost shareFolder]$ hdfs dfs -tail /user/hadoop/output/part-00000
2024-07-19 01:17:56,536 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
        4
"voulez"        14
"vouliez"       1
"voulions"      1
"vouloir"       4
"voulons"       1
"voulu" 16
"voulue"        1
"voulurent"     1
"voulus"        1
"voulut"        3
"vous"  371
"voyage"        4
"voyageant"     1
"voyages"       1
"voyageur"      2
"voyageurs"     1
"voyaient"      1
"voyait"        1
"voyance"       1
"voyant"        7
"voyelles"      1
"voyez" 12
"voyons"        5
"vrai"  19
"vraie" 2
"vraiment"      11
"vrais" 1
"vraisemblance" 1
"vre"   2
"vres"  4
"vrier" 8
"vu"    35
"vue"   10
"vues"  2
"vulgaires"     1
"vuln"  1
"vus"   2
"wace"  2
```

6. Conclusion

Ce TP a permis de mettre en place et d'exécuter un job MapReduce avec Hadoop et Python. Nous avons configuré Hadoop, préparé les données, écrit un programme MapReduce pour compter les occurrences des mots, et vérifié les résultats sur HDFS. Cette expérience a illustré les principes fondamentaux de MapReduce et le traitement parallèle de données massives.