TP2 :Traitement Batch avec Hadoop Streaming

Objectifs:

Ce TP a pour objectif de permettre une familiarisation avec l'utilisation de Hadoop Streaming pour implémenter des tâches de traitement de données via MapReduce en Python. Les participants développeront des programmes de Map et de Reduce en Python, puis exécuteront des jobs MapReduce à l'aide de Hadoop Streaming.

Etape de installation du Python sur tous les nœuds du cluster

Pour cela on va acceder a chaque node et executer la commande

```
sudo apt-get install -y python
```

Partie 2 : Exécution d’un job avec Hadoop Streaming

Dans cet exemple, nous explorerons le code du programme **WordCount**, qui permet de compter le nombre d'occurrences de chaque mot dans un ensemble de fichiers texte. Le programme repose sur deux fonctions principales : une fonction de map (mapper) et une fonction de reduce (reducer).

Mapper : mapperWC.py

Le programme **mapper** traite chaque ligne d'entrée en découpant les mots qu'elle contient, puis génère pour chaque mot un couple clé-valeur, où la clé correspond au mot et la

valeur est égale à 1. Ces couples clé-valeur sont ensuite écrits sur la sortie standard (STDOUT). Un délimiteur (par défaut, "\t") est utilisé entre la clé et la valeur afin d'assurer un traitement correct des paires lors de l'étape de Shuffle.

```
C:\Users\nourh\docker-hadoop>notepad mapperWC.py
```

File Edit View

```
#!/usr/bin/env python
"""mapper.py"""
import sys
for line in sys.stdin:
        line = line.strip()
        words = line.split()
        for word in words:
                print('{}\t{}'.format(word, 1))
```

Reducer: reducerWC.py

Le programme reducer agrège les paires clé-valeur en additionnant le nombre de fois que chaque mot
apparaît. Il renvoie le résultat sur STDOUT.

```
#!/usr/bin/env python
import sys
# Initialize variables
current_word = None
current_count = 0
# Iterate through input lines, which are sorted by key (word) in ascending order
for line in sys.stdin:
# Remove leading and trailing whitespace
        line = line.strip()
# Split the key (word) and value (count) by a tab character
        word, count = line.split('\t', 1)
# Convert the count to an integer
        try:
                count = int(count)
        except ValueError:
# If the conversion fails, skip this line
                continue
# If the current word is the same as the previous word, increment the count
        if current_word == word:
                current_count += count
        else:
# If the word changes, print the result for the previous word
                if current_word:
                        print('{}\t{}'.format(current_word, current_count))
# Reset the variables for the new word
                current_word = word
                current_count = count
# Print the result for the last word
if current_word == word:
        print('{}\t{}'.format(current_word, current_count))
```

Exécution du programme sur le cluster :

1. Transférer les scripts dans le cluster Hadoop :

```
C:\Users\nourh\docker-hadoop>docker cp mapperWC.py namenode:/mapperWC.py
Successfully copied 2.05kB to namenode:/mapperWC.py
```

```
C:\Users\nourh\docker-hadoop>docker cp reducerWC.py namenode:/reducerWC.py
Successfully copied 2.56kB to namenode:/reducerWC.py

C:\Users\nourh\docker-hadoop>
```

2. Entrer dans le container du namenode et tester les scripts :

```
C:\Users\nourh\docker-hadoop>docker exec -it namenode bash
root@7081d4288712:/# ls
KEYS  entrypoint.sh  hadoop-streaming-2.7.3.jar  lib          media  plz.txt     reducerWC.py  sbin  usr
bin   etc            home                        lib64        mnt    proc        root          srv   var
boot  hadoop         input                       mapper.py    opt    reduce.py   run           sys   words.txt
dev   hadoop-data    jar_files.zip               mapperWC.py  p.txt  reducer.py  run.sh        tmp
root@7081d4288712:/# chmod u+x mapperWC.py
root@7081d4288712:/# chmod u+x reducerWC.py
root@7081d4288712:/#
```

En premier lieu, vous allez tester votre code sur les petits fichiers suivants. Vous pouvez par la suite le tester sur les fichiers du dossier **Data** (créé dans le TP1.)

3. Créer des fichiers textes et les transférer dans HDFS :

```
root@7081d4288712:/# hdfs dfs -ls ./input2
Found 4 items
-rw-r--r--   3 root supergroup         12 2024-11-29 10:24 input2/f1.txt
-rw-r--r--   3 root supergroup         13 2024-11-29 10:24 input2/f2.txt
-rw-r--r--   3 root supergroup         13 2024-11-29 10:24 input2/f3.txt
-rw-r--r--   3 root supergroup         16 2024-11-29 10:24 input2/f4.txt
root@7081d4288712:/# hdfs dfs -cat ./input2/f2.txt
2024-11-29 10:26:04,891 INFO sasl.SaslDataTransferClient: SASL encryption tr
ust check: localHostTrusted = false, remoteHostTrusted = false
Hello Docker
root@7081d4288712:/#
```

```
root@7081d4288712:/# mkdir input2
root@7081d4288712:/# echo "Hello World" > input2/f1.txt
root@7081d4288712:/# echo "Hello Docker" > input2/f2.txt
root@7081d4288712:/# echo "Hello Hadoop" > input2/f3.txt
root@7081d4288712:/# echo "Hello MapReduce" > input2/f4.txt
root@7081d4288712:/# hadoop fs -mkdir -p input2
root@7081d4288712:/# hdfs dfs -put ./input2/* input2
```

4. Exécuter le programme MapReduce

```
root@7081d4288712:/# hadoop jar /opt/hadoop-3.2.1/share/hadoop/tools/lib/had
oop-streaming-3.2.1.jar \
> -files mapperWC.py,reducerWC.py \
> -input input2 \
> -output output \
> -mapper mapperWC.py \
> -reducer reducerWC.py
packageJobJar: [/tmp/hadoop-unjar6545346900417680726/] [] /tmp/streamjob3252
435089132406727.jar tmpDir=null
2024-11-29 10:34:19,407 INFO client.RMProxy: Connecting to ResourceManager a
t resourcemanager/172.18.0.4:8032
2024-11-29 10:34:19,528 INFO client.AHSProxy: Connecting to Application Hist
ory server at historyserver/172.18.0.3:10200
2024-11-29 10:34:19,554 INFO client.RMProxy: Connecting to ResourceManager a
t resourcemanager/172.18.0.4:8032
2024-11-29 10:34:19,555 INFO client.AHSProxy: Connecting to Application Hist
ory server at historyserver/172.18.0.3:10200
2024-11-29 10:34:19,730 INFO mapreduce.JobResourceUploader: Disabling Erasur
e Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1732873976123_
0001
2024-11-29 10:34:19,830 INFO sasl.SaslDataTransferClient: SASL encryption tr
ust check: localHostTrusted = false, remoteHostTrusted = false
2024-11-29 10:34:19,903 INFO sasl.SaslDataTransferClient: SASL encryption tr
ust check: localHostTrusted = false, remoteHostTrusted = false
2024-11-29 10:34:19,927 INFO sasl.SaslDataTransferClient: SASL encryption tr
ust check: localHostTrusted = false, remoteHostTrusted = false
```

5. Voir les résultats :

```
hdfs dfs -ls output
```

```
root@7081d4288712:/# hdfs dfs -ls output
Found 2 items
-rw-r--r--   3 root supergroup          0 2024-11-29 10:38 output/_SUCCESS
-rw-r--r--   3 root supergroup         30 2024-11-29 10:38 output/part-r-000
00
```

Conclusion

Ce TP a permis de découvrir l'utilisation d'Hadoop Streaming pour exécuter des tâches MapReduce en Python. Nous avons implémenté et testé les fonctions mapper et reducer, puis analysé les résultats dans HDFS. Cette approche constitue une base pour traiter efficacement de grandes quantités de données distribuées.