TP n° 2 Spark

SAULNIER Guillaume binôme 1

PUILLANDRE Valentin binôme 2

I. Spark et Hadoop

Relancer les containers dockers que vous avez : docker start hadoop-master hadoop-worker1 hadoop-worker2

Aller sur le container master : docker exec -it hadoop-master bash puis lancer hadoop : ./start-hadoop.sh

Verifier que les processus background sont bien lancés avec la commande : jps.

Quels sont les processus qui tournent sur le container master ?

```
C:\Users\valen>docker exec -it hadoop-master bash
root@hadoop-master:~# jps
468 SecondaryNameNode
199 NameNode
761 ResourceManager
1178 Jps
```

Faire de même avec les workers : docker exec -it hadoop-worker1 bash. Quels sont les processus qui tournent sur le worker ?

```
root@hadoop-worker1:~# jps
704 SecondaryNameNode
214 NodeManager
74 DataNode
1979 Jps
```

II. Premier pas avec Spark

Créer un fichier dans le container master nommé fichier1.txt avec le contenu suivant :

Salut salut Spark ! On compte les mots.

Salut salut Hadoop ! On va compter les mots.

Mettre ce fichier dans le dossier input sur le HDFS.

```
root@hadoop-master:~# hdfs dfs -ls input
Found 2 items
-rw-r--r--   2 root supergroup         85 2024-05-24 08:33 input/fichier1.tx
t
-rw-r--r--   2 root supergroup  211312924 2024-04-04 10:25 input/purchases.t
xt
```

Pour lancer spark, il faut taper la commande : spark-shell. Quel est le langage utilisé ici ?

```
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 1.8.0_392)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Faire un MapReduce pour compter les mots du fichier1.txt que vous avez crée en utilisant Scala. Vous pouvez vous appuyer sur ChatGPT pour générer ce code. Avant de l'exécuter, merci de m'appeler pour venir vérifier. (4 lignes suffissent normalement).

```
val textFile = sc.textFile("hdfs:///user/root/input/fichier1.txt")

val counts = textFile.flatMap(line => line.split("\\s+")).map(word => (word, 1)).reduceByKey(_ + _)


counts.saveAsTextFile("hdfs:///user/root/output/wordcount")
counts.collect().foreach(println)
```

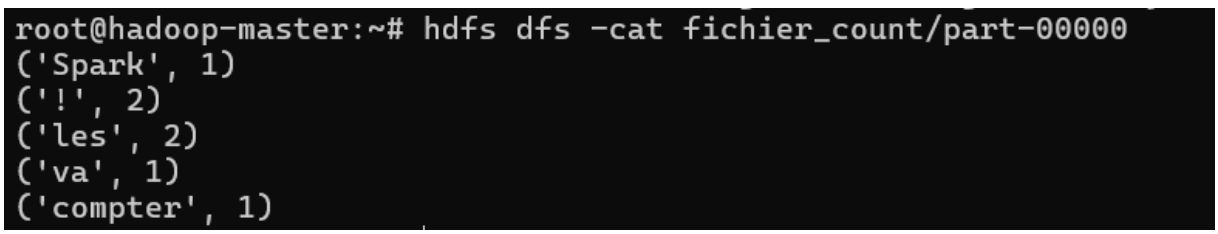
III. Lancer un wordcount à travers python et Spark

Fichier Python pour lancer un wordcount sur le fichier demandé via spark.

```
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("WordCount") \
    .getOrCreate()

# Load the text file
lines = spark.sparkContext.textFile("fichier1.txt")

# Split each line into words
words = lines.flatMap(lambda line: line.split())

# Map each word to a tuple (word, 1) and then reduce by key to count occurrences
word_counts = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

# Save the word counts to a text file
word_counts.saveAsTextFile("fichier_count")

# Stop the SparkSession
spark.stop()
```

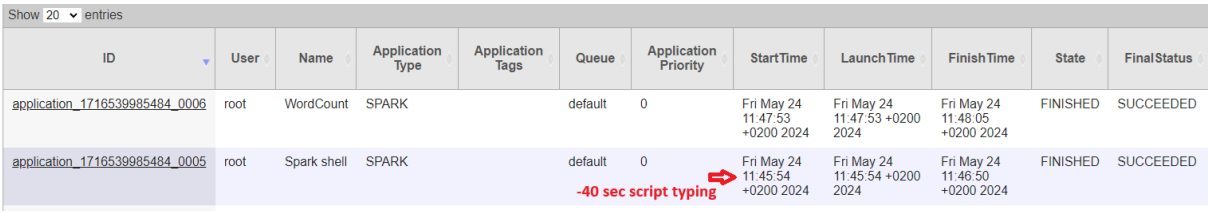
Pour lancer le code : spark-submit word_count.py

Récupérer les fichiers de sorties crées et vérifier la bonne exécution du wordcount. Insérer ici les screenshots des résultats.



Regarder sur http://localhost:8088/ l'état des jobs et le temps d'exécution. Commenter en comparant par rapport à celui pris par un MapReduce vu lors du dernier TP.

Le temps d'exécution par rapport au TP précédent est beaucoup plus rapide. Sinon entre scala et python, le temps d'exécution et quasi le même.



Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus
application_1716539985484_0006	root	WordCount	SPARK		default	0	Fri May 24 11:47:53 +0200 2024	Fri May 24 11:47:53 +0200 2024	Fri May 24 11:48:05 +0200 2024	FINISHED	SUCCEEDED
application_1716539985484_0005	root	Spark shell	SPARK		default	0	Fri May 24 11:45:54 +0200 2024	Fri May 24 11:45:54 +0200 2024	Fri May 24 11:46:50 +0200 2024	FINISHED	SUCCEEDED

Comme précisé en cours, il y a deux modes pour lancer le code avec spark submit sur le cluster ou en local sur votre machine.

Quels sont les deux commandes différentes pour lancer le code en local et sur le container docker ? Tester les deux commandes. Est-ce que vous voyez un changement de vitesse d'exécution ?

spark-submit word_count.py dans le container docker,

IV. SparkStreaming

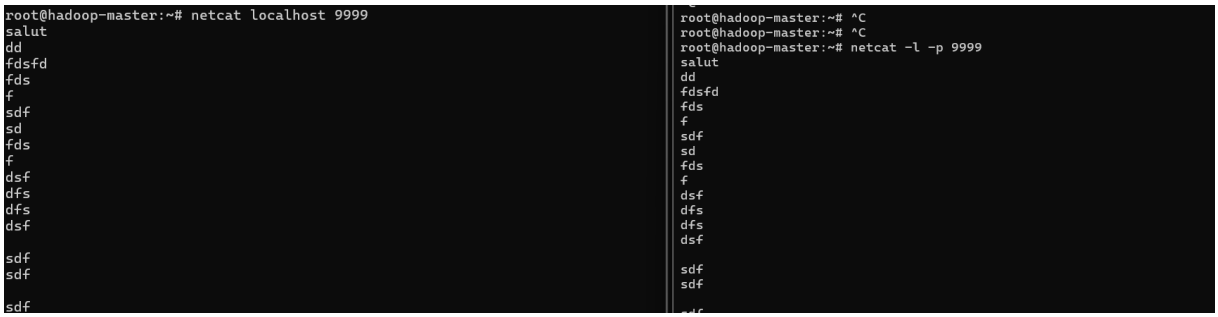
On va maintenant utiliser une autre fonctionnalité de Spark, celle de la gestion de flux de données. Nous allons lancer un « chat » dans notre container pour lui envoyer des messages et intercepter ces messages pour les process en temps réel avec Spark.

Dans un premier temps configurer un chat sur deux terminal en mode envoie et ecoute sur le port 9999 à l'aide de netcat.

Le code script python pour configurer hadoop-streaming :

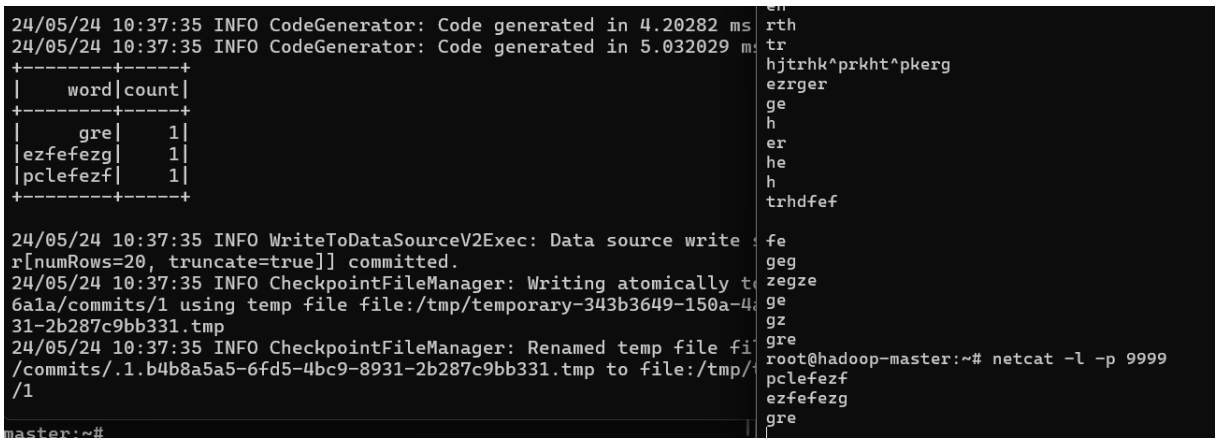
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split

# Initialisation de la session Spark
spark = SparkSession \
    .builder \
    .appName("NetworkWordCount") \
    .getOrCreate()

# Configuration de la lecture en continu à partir du socket localhost:9999
lines = spark \
    .readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 9999) \
    .load()

# Diviser les lignes en mots
words = lines.select(
    explode(split(lines.value, " ")).alias("word")
)
```

```
# Compter le nombre d'occurrences de chaque mot
wordCounts = words.groupBy("word").count()

# Écrire les résultats continus sur la console, rafraîchissant chaque seconde
query = wordCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .trigger(processingTime="1 second") \
    .start()

# Attendre la terminaison du traitement en continu
query.awaitTermination()
```



Pour lancer le code : spark-submit streaming_word_count.py

Où s'execute hadoopstreaming ? Parler de l'intervalle de temps pour le traitement des données.



Il s'exécute dans le hadoop master, le traitement des données se fait en direct.

V. Bonus Kafka

Lancer kafka à l'aide de ./start-kafka-zookeeper.sh vérifier que le job s'exécute bien. Créer un tropic puis un producer et un consumer et envoyer des messages pour voir leurs retours.

Command list

Commandes & Étapes	Commentaires
// open docker desktop //	
docker start hadoop-master hadoop-worker1 hadoop-worker2	Lance workers1/2/master
docker exec -it hadoop-master bash	connect to master
./start-hadoop.sh	Lance hadoop
jps	Vérification process
exit	sors du master
docker exec -it hadoop-worker1 bash	connect to worker1
./start-hadoop.Sh	lance hadoop
jps	
exit	
docker exec -it hadoop-master bash	connect to master
./start-hadoop.sh	
nano fichier1.txt	on crée le fichier à tester
cat fichier1.txt	on vérifie le contenu parce qu'on est pas doué avec les commande linux
hdfs dfs -put fichier1.txt input/fichier1.txt	copy file to hadoop
// error copying //	
// put: Cannot create file/user/root/input/fichier1.txt.*COPYING*. Name node is in safe //	
hdfs dfsadmin -safemode leave	leaving safe mode
hdfs dfs -put fichier1.txt input/fichier1.txt	copy file to hadoop
spark-shell	démare spark-scala
language: scala	
// création d'un mapReduce avec chatgpt en scala (yikes) //	
// wordcount ok //	
hdfs dfs -cat output/part-00000	

hdfs dfs -cat output/part-00001	
nano word_count.py	
// copy script in tp2 //	
// replace "fichier1.txt" => "hdfs:///user/root/input/fichier1.txt" //	
spark-submit word_count.py	
hdfs dfs -cat fichier_count/part-00000	
hdfs dfs -cat fichier_count/part-00001	
// no significant difference between python & scala on 200mb file //	
apt-get install netcat	install netcat
nc -l -p 9999	netcat listen port 9999
// ouvre second terminal docker hadoop-master //	
nc localhost 9999	ouvre second terminal docker hadoop-master
// ouvre un 3ième terminal docker hadoop-master //	
nano streaming_word_count.py	
// copy/paste code + change 1 second to 10 seconds //	
cat streaming_word_count.py	
spark-submit streaming_word_count.py	
./start-kafka-zookeeper.sh	