COLD2 – TP 01 – Hadoop, Spark, Kafka et HBase

Matériel : docker

Il vous est demandé un compte-rendu (en PDF) précisant les commandes et les résultats (capture de texte principalement).

1. Démarrage du cluster Hadoop :

a) Récupérer l'image Hadoop sur le registre docker de l'IUT :

```
docker pull registry.iutbeziers.fr/cb_hadoop:3.2.1
```

b) Créer un pont réseau docker pour le cluster Hadoop :

```
docker network create --driver=bridge hadoop
```

c) Lancer les conteneurs docker Hadoop à l'aide du script fourni :

```
sh start-hadoop-cluster-docker.sh
```

d) Exécuter enfin le script de démarrage du cluster Hadoop.

```
root@hadoop-master:~# start-all.sh
```

e) Afficher alors la liste des processus JAVA en fonctionnement sur `hadoop-master` avec la commande `jps`.

f) Afficher ensuite l'état du système HDFS et noter le nom des 2 machines esclaves du cluster :

```
hdfs dfsadmin -report
```

g) Comparer aux informations disponibles sur l'URL http://localhost:9870/.

h) Comment modifier la machine locale pour que l'on puisse cliquer sur les liens de la page « Datanodes » (http://hadoop-slave1:9864 et http://hadoop-slave2:9864) ?

i) Créer ensuite le répertoire de base pour l'utilisateur root :

```
hadoop fs -mkdir -p /user/root
```

j) Tester alors les options `-ls`, `-put`, `-get`, `-mv`, `-rm`, `-cat`, `-tail` de la commande `hadoop fs`.

k) Tester les différents menus de la version WEB et notamment « Utilities » et « Browse the filsystem ».

2. Premier test de Map/Reduce

a) Créer le répertoire `data_in` sur HDFS et y copier les fichiers textes contenus dans l'archive `files.tar.gz` fournie (Utiliser `docker cp` pour transférer ce fichier de votre machine vers le conteneur docker).

b) Lancer ensuite les commandes suivantes :

```
JAR=$HADOOP_HOME/share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-3.2.0-sources.jar
hadoop jar $JAR org.apache.hadoop.examples.WordCount data_in data_out
```

c) Afficher enfin le contenu du répertoire `data_out` et vérifier que le programme a bien compté le nombre d'occurrences de chaque mot de tous les fichiers sources.

d) Visualiser sur le site WEB http://localhost:8088/, les détails de l'application Map/Reduce qui vient d'être soumise au cluster.

3. Hadoop Streaming (Map/Reduce en python)

La librairie Hadoop Streaming permet de créer des taches Map/Reduce avec n'importe quel exécutable.

a) Il est alors possible d'écrire les étapes Map et Reduce avec des programmes Python (voir fichiers `mapper.py` et `reducer.py`). On peut en premier lieu, tester ces programmes en local ainsi :

```
head -50 file1.txt | ./mapper.py | sort |./reducer.py
```

b) Si cela fonctionne bien, on peut lancer l'exécution sur le cluster Hadoop :

```
JAR=$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.2.0.jar
hadoop jar $JAR -files mapper.py,reducer.py -mapper mapper.py -reducer reducer.py
  -input data_in -output data_out2
```

4. Apache spark :

Apache spark permet d'accélérer les temps de calcul des opérations Map/Reduce. On peut en premier lieu, tester en local les programmes puis les faire évaluer ensuite sur le cluster.

a) Si on veut programmer en Python, on peut tester les instructions sur un interpréteur local :

```
pyspark
```

b) Afficher alors le contenu de la variable `sc` pour vérifier que l'on se trouve bien dans un contexte local.

```
print(sc)
```

c) Créer un premier RDD à partir d'un fichier texte local et vérifier son type et son contenu.

```
lines=sc.textFile('file:///root/file1.txt')
print(lines)
lines.collect()
```

d) Découper les lignes pour obtenir un RDD contenant tous les mots de toutes les lignes :

```
words=lines.flatMap(lambda x: x.split())
words.collect()
```

e) Créer un RDD avec des couples (mot,1) :

```
tuples=words.map(lambda x: (x, 1))
tuples.collect()
```

f) Réaliser l'opération de réduction en sommant les valeurs pour chaque mot :

```
res=tuples.reduceByKey(lambda a,b: a+b)
res.collect()
```

g) Trier enfin les valeurs dans l'ordre décroissant et ne conserver que les 5 premiers :

```
mostUsed=sc.parallelize([res.takeOrdered(5, key = lambda x: -x[1])])
mostUsed.collect()
```

NB : Pour l'ordre croissant des valeurs, indiquer : `lambda x: x[1]`. Si on veut trier les mots dans l'ordre alphabétique, il faut utiliser `lambda x: x[0]` et `lambda x: -x[0]` pour l'ordre inverse.

h) Enfin, on peut sauvegarder les résultats en local :

```
mostUsed.saveAsTextFile('file:///root/data_out3')
```

i) Récapituler toutes les commandes dans un script Python en créant un contexte au tout début :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pyspark import SparkContext
sc=SparkContext(appName="WordStats")
...
```

j) Lancer enfin le script avec `spark-submit` :

```
spark-submit wordStats.py
```

k) Pour réaliser la même chose sur les fichiers du cluster HDFS, il faut changer les URL des fichiers : `file:///root` devient `hdfs:///user/root`. Ensuite on indique au système d'utiliser YARN :

```
spark-submit --master yarn wordStatsHdfs.py
```

l) Vérifier ensuite sur le site WEB:8088, le détail des applications soumises au cluster.

5. Traitement de données issues d'un routeur :

Un outils de « monitoring » réseau sauvegarde en permanence toutes les minutes, un fichier texte contenant les informations suivantes (voir un extrait dans `00-00-head10.csv`) :

```
SRC_MAC,DST_MAC,SRC_IP,DST_IP,SRC_PORT,DST_PORT,PROTOCOL,TOS,PACKETS,BYTES
50:9a:4c:85:79:2c,70:77:81:46:98:48,172.217.18.225,172.31.1.34,443,61109,tcp,32,9141,13427332
70:77:81:46:98:48,50:9a:4c:85:79:2c,172.31.1.34,172.217.18.225,61109,443,tcp,0,4636,185644
b0:34:95:f3:94:aa,50:9a:4c:85:79:2c,172.31.1.35,194.199.227.201,54863,22,tcp,0,2961,4180800
50:9a:4c:85:79:2c,b0:34:95:f3:94:aa,194.199.227.201,172.31.1.35,22,54863,tcp,8,5721,367012
50:9a:4c:85:79:2c,c8:14:51:7c:4a:7e,54.85.177.93,172.31.0.85,443,33033,tcp,40,6,903
c8:14:51:7c:4a:7e,50:9a:4c:85:79:2c,172.31.0.85,54.85.177.93,33033,443,tcp,0,3,574
b0:34:95:f3:94:aa,50:9a:4c:85:79:2c,172.31.1.35,194.199.227.201,54862,22,tcp,0,3713,5141388
50:9a:4c:85:79:2c,b0:34:95:f3:94:aa,194.199.227.201,172.31.1.35,22,54862,tcp,8,7197,450508
b0:34:95:f3:94:aa,50:9a:4c:85:79:2c,172.31.1.35,172.217.18.225,54867,443,tcp,0,2136,141580
```

On veux pouvoir utiliser Spark pour préparer des synthèses statistiques sachant que le réseau interne est en `172.31.0.0/16`. On commencera par écrire des petits scripts agissant sur un fichier donné puis ensuite sur tous les fichiers d'une journée (voir `2019-01-25.tar.xz`).

a) Écrire un petit script python Spark qui détermine le nombre total de paquets envoyés ou reçus par le routeur. On pourra chercher en premier comment lire un fichier CSV avec pySpark pour obtenir un objet pyspark.sql.DataFrame... Puis consulter la documentation en ligne : https://spark.apache.org/docs/latest/api/python/index.html.

b) Écrire ensuite un petit script python Spark qui détermine le volume total de données entrantes et sortantes. On pourra faire un filtre sur les colonnes de l'objet DataFrame précédent.

c) Comment connaître les ports les plus utilisés ?

d) Comment trouver l'adresse MAC la plus utilisée en volume ?

6. Apache kafka :

Apache kafka est un système de messaging (publisher/suscriber) ou les données sont stockées dans des « topics » découpés en « partitions ». Il est utilisé en général pour récolter les données que l'on traitera ensuite avec Spark Streaming.

e) Démarrer le serveur Zookeeper et un serveur kafka :

```
zookeeper-server-start.sh -daemon $KAFKA_HOME/config/zookeeper.properties
kafka-server-start.sh -daemon $KAFKA_HOME/config/server-master.properties
```

f) Vérifier que l'on voit bien les processus JAVA `QuorumPeerMain` et `Kafka` avec `jps`.

g) Créer un premier sujet (topic) :

```
kafka-topics.sh --create --zookeeper hadoop-master:2181
  --replication-factor 1 --partitions 1 --topic XXXXX
```

h) Vérifier qu'il apparaît bien dans la liste des sujets du serveur zookeper :

```
kafka-topics.sh --list --zookeeper hadoop-master:2181
```

i) Essayer de créer un second sujet avec un facteur de réplication de 2. Quel message apparaît ?

j) Essayer de créer un nouveau sujet avec un nombre de partitions de 2 puis utiliser l'option `--describe` pour afficher ses propriétés.

k) Comment effacer un sujet de la liste ?

l) Créer maintenant un producteur de données dans la console courante sur un sujet existant :

```
kafka-console-producer.sh --broker-list hadoop-master:9092
  --topic XXXXX
```

m) Ouvrir une seconde console sur le conteneur `hadoop-master` et lancer un consommateur de données sur le même sujet que la question précédente :

```
docker exec -it hadoop-master bash
root@hadoop-master:~#
kafka-console-consumer.sh --bootstrap-server hadoop-master:9092
  --topic XXXXX
```

n) Taper enfin plusieurs lignes de texte dans la première console (producteur) et vérifier qu'il apparaît automatiquement dans la seconde (consommateur).

o) Tester d'ouvrir une troisième console et vérifier que l'on peut bien récupérer tous les messages postés sur le sujet depuis le début avec l'option `--from-beginning`.

p) Vérifier que le module `kafka-python` est bien présent en affichant avec `pip`, la liste des paquets installés sur le conteneur docker `hadoop-master`.

q) Tester alors les scripts `consumer.py` et `producer.py` en adaptant le sujet à utiliser (topic).

r) On peut se connecter en direct sur un « broker » Kafka pour récupérer les données et les traiter dans un script Python. Adapter alors le fichier d'exemple fourni dans Apache Spark `$SPARK_HOME/examples/src/main/python/streaming/direct_kafka_wordcount.py` en modifiant la durée de traitement du `StreamingContext` à 20 secondes et en fixant le niveau de debug à WARN pour ne pas trop « polluer » la console.

```
...
sc = SparkContext(appName="PythonStreamingDirectKafkaWordCount")
ssc = StreamingContext(sc, 20)
sc.setLogLevel("WARN")
...
```

Exécuter enfin ce script avec la commande suivante en adaptant la valeur du sujet (topic) :

```
spark-submit --packages
   org.apache.spark:spark-streaming-kafka-0-8_2.11:2.4.0
   direct_kafka_wordcount.py hadoop-master:9092 XXXXX
```

NB : Voir https://spark.apache.org/docs/2.4.0/streaming-kafka-integration.html pour comprendre la syntaxe utilisée différente de celle indiquée dans le commentaire du script python.

s) On peut aussi sauvegarder en plusieurs fichiers les différents résultats en ajoutant l'instruction :

```
counts.saveAsTextFiles('file:///root/kafka_out')
```

t) Vérifier que l'on retrouve bien les données dans les fichiers générés. Utiliser `cat` pour afficher la concaténation de tous les fichiers d'un répertoire donné.

u) Si on veut plutôt utiliser le serveur Zookeper, il faut utiliser l'exemple `kafka_wordcount.py`. Tester le script après l'avoir modifié comme précédemment.

v) On veut maintenant tester la réplication des topics sur plusieurs « brokers ». Démarrer sur chaque esclave un serveur Kafka (Noter la valeur `broker.id` de chaque fichier `properties`) :

```
docker exec -it hadoop-slave1 bash
root@hadoop-slave1:~#
kafka-server-start.sh -daemon $KAFKA_HOME/config/server-slave1.properties

docker exec -it hadoop-slave2 bash
root@hadoop-slave2:~#
kafka-server-start.sh -daemon $KAFKA_HOME/config/server-slave2.properties
```

w) Vérifier ensuite que le serveur Zookeper voit bien les id des 3 brokers.
`zookeeper-shell.sh hadoop-master:2181 <<< "ls /brokers/ids"`

x) Créer alors un sujet avec un facteur de réplication de 3 puis afficher la description du sujet pour noter l'identifiant du « Leader » et les « Replicas ».

y) Connecter un « producer » et un « consumer » sur `haddop-slave1` et envoyer des données.

z) Arrêter alors le serveur `hadoop-master`.

```
root@hadoop-master:~# kafka-server-stop.sh
```

aa) Quels messages apparaissent dans la console du « consumer » ?

bb) Peut-il recevoir de nouveaux messages envoyés par le « producer » ?

cc) Consulter à nouveau la description du sujet pour noter les différences par rapport à la question 6.x.

dd) Que se passe-t-il quand le serveur `hadoop-master` est relancé ?

7. Apache HBase :

Apache HBase est une base noSQL orientée colonne créé pour s'intégrer sur un système HDFS.