Travaux pratiques préparatoires : Cassandra

Jonathan Lejeune

Objectifs

Ce sujet de travaux pratiques vous indique comment installer Cassandra et le driver scala Spark-Cassandra dans un environnement UNIX.

Pré-requis

Vous devez avoir installé Java 8 ou supérieur (tapez `java -version`) et python 2.7 (tapez `python -version`) pour pouvoir utiliser cqlsh. Pour le driver Spark-Cassandra, il est nécessaire d'avoir installé :

- git
- scala 2.12
- Spark 3 (voir TME d'installation de Spark).

Introduction

Cassandra, développé par la fondation Apache, est un SGBD NoSQL orienté document et basé sur une architecture distribuée pair à pair. Il permet aux utilisateurs de paramétrer leurs requêtes pour exprimer un niveau de cohérence sur le résultat attendu. Contrairement à la plupart des systèmes distribués Apache (Hadoop, Spark, Hbase, ...), Cassandra ne possède pas nativement un mode pseudo-distribué permettant d'émuler un ensemble de nœuds distants dans une seule machine hôte (notamment à cause des ports d'écoutes qui sont constants pour tous les nœuds du cluster Cassandra). Si on l'on souhaite tester Cassandra en mode pair à pair il est donc nécessaire d'avoir soit plusieurs machines hôtes (physiques et/ou virtuelles) soit utiliser des mécanismes de conteneurisation type Docker.

Exercice 1 – Installation et démarrage de Cassandra

Question 1

Téléchargez la dernière version de Cassandra à l'URL suivante.

`http://apache.mediamirrors.org/cassandra/`

. Extrayez le contenu de l'archive à l'endroit souhaité dans votre home (attention aux quotas).

Question 2

Au même titre qu'Hadoop, Cassandra a besoin que la variable d'environnement JAVA_HOME soit définie. **Si ce n'est pas déjà fait** ajoutez la ligne suivante dans votre *.bashrc* :

```
export JAVA_HOME=<chemin vers la JVM courante de votre machine>
```

Ajouter ensuite les lignes suivantes :

```
export MY_CASSANDRA_HOME=<votre_repertoire_installation_cassandra>
export PATH=$PATH:$MY_CASSANDRA_HOME/bin
```

Pour prendre en compte ces modifications dans votre terminal ouvert tapez :

```
source ~/.bashrc
```

Question 3

Un serveur cassandra a besoin des ports réseau suivants pour fonctionner. Leurs valeurs par défaut sont 9042, 7000, 7199 et 46185. Vérifiez grâce à la commande `netstat -na | grep LISTEN | grep tcp`. Si un de ces ports est déjà utilisé, il est possible de changer la valeur du port d'écoute dans le fichier `$MY_CASSANDRA_HOME/conf/cassandra.yaml`.

Question 4

Cassandra utilise 4 dossiers sur le système de fichiers de chaque nœud : (1) un dossier pour les données, (2) un dossier pour la journalisation des requêtes (commitlog), (3) un dossier pour des fichiers de cache et (4) un dossier de méta données (hints). En phase de test, on préférera configurer Cassandra pour que ces dossiers soit stockés dans le `/tmp` de la ou des machine(s) du cluster. Pour cela, éditez le fichier `$MY_CASSANDRA_HOME/conf/cassandra.yaml` et modifiez les propriétés suivantes (faire une recherche de texte via votre éditeur de texte) :

```
data_file_directories:
     - /tmp/cassandra/data
```

(attention de bien respecter pour ce paramètre le retour chariot, le retrait et le caractère -)

```
commitlog_directory: /tmp/cassandra/commitlog
saved_caches_directory: /tmp/cassandra/saved_caches
hints_directory: /tmp/cassandra/hints
```

Attention : Avec cette configuration, les données stockées dans Cassandra risquent d'être effacées si votre système d'exploitation nettoie le dossier `/tmp` à chaque redémarrage.

Question 5

La commande `cassandra` aura pour effet de démarrer un nœud Cassandra en arrière plan sur votre machine. Pour l'arrêter vous devez passer par la commande `kill`. Si vous souhaitez lancer le serveur en avant plan, il faut ajouter l'option `-f`. Ceci aura pour effet de garder la main sur votre invite de commande, et il faudra faire un Ctrl-C pour l'arrêter.

Question 6

Il est possible d'administrer le cluster cassandra avec la commande `nodetool`

Question 7

Pour lancer un client CQL sur un serveur Cassandra, tapez la commande

```
cqlsh <adresse_serveur>
```

Par défaut le serveur sur lequel on se connecte est `localhost`.
Il est possible que cette commande vous renvoie une erreur ressemblant à ceci :

```
Connection error: ('Unable to connect to any servers',
{'127.0.0.1': ProtocolError('Unexpected response during Connection
 setup: AttributeError("\'module\' object has no attribute \'decompress\'",)',)})
```

Ce problème vient d'un bug lors de l'utilisation de package de compression python (lz4 ou snappy) qui est activée par défaut si un de ces packages est installé. La solution est de désactiver la compression lorsque le client établit la connexion avec le serveur. Malheureusement il n'est pas connu à ce jour un moyen de faire cette configuration autre que de modifier le script python lui même. Pour cela :

- ouvrez un éditeur de texte sur le script *cqlsh.py* qui se trouve dans le répertoire *bin* de votre installation Cassandra.
- cherchez une ligne qui contient `self.conn = Cluster` (environ ligne 481 pour Cassandra 3.11.5).
- Ajouter dans l'appel au constructeur `Cluster`, le paramètre `compression=False` (n'oubliez pas la virgule ensuite)
- Vérifiez que la commande `cqlsh` fonctionne

Question 8

Dans le client CQL créez un keyspace `test` et une table `kv` :

```
CREATE KEYSPACE test
    WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1 };
CREATE TABLE test.kv(key text PRIMARY KEY, value int);
```

N.B. : il vous est possible d'utiliser la touche `tab` pour l'auto complétion des commandes CQL qui permet de faciliter grandement leur saisie..

Question 9

Insérez deux lignes :

```
INSERT INTO test.kv(key, value) VALUES ('key1', 1);
INSERT INTO test.kv(key, value) VALUES ('key2', 2);
```

Question 10

Listez la table :

```
SELECT * FROM test.kv;
```

N.B. : Il vous est possible également de prendre le keyspace `test` par défaut avec la commande `USE test;` vous évitant ainsi de préfixer le nom de la table par celui-ci.

Question 11

Pour quitter le shell CQL, tapez `quit`.

Exercice 2 – Installation et test du driver Spark-Cassandra

Cet exercice vous indique comment installer le driver Spark-Cassandra (développé par Datastax). L'installation de ce driver passe par une compilation des sources qui sont disponibles sur le dépôt GitHub de Datastax. La compilation se fera à travers `sbt` qui est un utilitaire de compilation pour scala et java.

Question 1

Téléchargez la dernière version de `sbt` à l'URL suivante (nous prendrons l'archive zip ici) :

```
https://www.scala-sbt.org/download.html
```

Question 2

Dézipper l'archive téléchargée ci-avant et ajouter dans votre .bashrc les lignes suivantes :

```
export SBT_HOME=<chemin vers l'installation sbt>
export PATH=$PATH:$SBT_HOME/bin
```

et éventuellement faire `source $HOME/.bashrc` pour prendre en compte ces modifications dans votre terminal ouvert.

N.B. : si vous êtes root sur votre machine, il est également possible d'installer SBT via le gestionnaire de paquet de votre distribution Linux.

Question 3

Nous allons maintenant cloner le dépôt git du driver Spark-Cassandra. En vous assurant le cas échéant que les paramètres de proxy de git sont correctement configurés, tapez :

```
git clone https://github.com/datastax/spark-cassandra-connector.git
```

Question 4

Placez vous à la racine du dossier téléchargé ci-avant et tapez les commandes suivantes (ceci peut prendre quelques minutes) :

```
sbt -Dscala-2.12=true doc
sbt -Dscala-2.12=true package
sbt -Dscala-2.12=true assembly
```

Si vous devez passer par un proxy, ajoutez pour chaque commande les options suivantes :

```
-Dhttps.proxyHost=<name_proxy>
-Dhttps.proxyPort=<port_proxy>
-Dhttp.proxyHost=<name_proxy>
-Dhttp.proxyPort=<port_proxy>
```

À la fin de l'exécution de ces commandes, une archive `spark-cassandra-connector-assembly-<version>.jar` jar a du être produite dans le dossier `spark-cassandra-connector/connector/target/scala-2.12/`.

Question 5

En vous assurant que le serveur Cassandra démarré dans le précédent exercice soit toujours actif, ouvrez un spark-shell en spécifiant le jar créé à la question 4 :

```
spark-shell --jars /chemin/vers/spark-cassandra-connector-assembly.jar
```

Question 6

Pour utiliser le driver Spark-Cassandra importez les packages suivants :

```
import com.datastax.spark.connector._
import org.apache.spark.sql.cassandra._
```

Question 7

Pour accéder à une table depuis un sparkcontext, utilisez la méthode `cassandraTable` qui renvoie un `RDD[CassandraRow]` (voir la documentation pour plus d'information sur le type `CassandraRow`). Voici un exemple de programme qui affiche le nombre d'éléments dans la table `kv`, affiche le premier élément et affiche la somme de toutes les valeurs de la colonne `value`.

```
val rdd = sc.cassandraTable("test", "kv")
println(rdd.count)
println(rdd.first)
println(rdd.map(_.getInt("value")).reduce(_+_))
```

Testez-le dans votre spark-shell assurez-vous que les affichages sont cohérents.

Question 8

Pour écrire un rdd dans une table Cassandra, il faut utiliser la méthode `saveToCassandra`. Par exemple :

```
val rdd2 = sc.parallelize(Seq(("key3", 3), ("key4", 4)))
rdd2.saveToCassandra("test", "kv", SomeColumns("key", "value"))
```

Affichez le contenu de la table depuis un `cqlsh` et assurez-vous que la table a bien été modifiée.

Question 9

Pour utiliser le driver Spark-Cassandra sous Eclipse, vous devez :

- ajouter dans votre BuildPath de votre projet Eclipse le fichier jar généré à la question 4 (en plus des jars de Spark).
- ajouter dans la configuration de l'application Spark l'adresse d'un pair Cassandra avec la property `spark.cassandra.connection.host`

Si on devait exécuter le programme précédent, on écrirait le code suivant :

```
import org.apache.spark._
import com.datastax.spark.connector._
import org.apache.spark.sql._

object Essai extends App {
   org.apache.log4j.Logger.getLogger("org.apache.spark")
                          .setLevel(org.apache.log4j.Level.OFF)
   val conf = new SparkConf()
                    .setAppName("Spark␣on␣Cassandra").setMaster("local[*]")
                    .set("spark.cassandra.connection.host","localhost")
   val sc = new SparkContext(conf)
   val rdd = sc.cassandraTable("test", "kv")
   println(rdd.count)
   println(rdd.first)
   println(rdd.map(_.getInt("value")).reduce(_+_))

   val rdd2 = sc.parallelize(Seq(("key3", 3), ("key4", 4)))
   rdd2.saveToCassandra("test", "kv", SomeColumns("key", "value"))

   sc.stop()
}
```