Installation d'un cluster Apache Spark sous CentOS 7

28 août 2018

1 Objectifs

Apache Spark est une pateforme de traitement et d'analyse de gros volumes de données. Spark peut être utilisé sur un seul serveur ou sur un cluster constitué de plusieurs serveurs. Lorsque Spark est distribué sur un cluster, son architecture se divise en deux grandes parties : un serveur master (master node) et plusieurs serveurs esclaves (worker nodes). Le master node est constitué d'un driver program qui coordonne le traitement des processus en rapport avec une ou plusieurs applications et d'un cluster manager qui coordonne l'utilisation des ressources disponibles dans les worker nodes. Les machines virtuelles Java (encore appelées executors) sont créées dans les workers nodes par le driver program pour traiter les tâches en rapport avec une application donnée.

En cas de panne de certains executors et éventuellement d'un worker node, d'autres executors sont automatiquement crées (éventuellement) dans d'autres worker nodes pour reprendre les traitements alloués aux executors défaillants. Ces basculements se font sans aucune interruption des applications en cours de fonctionnement. Cette situation signifie pour le cluster que les ressources (CPU, RAM, disque dur) ont tout simplement diminuées. Par contre, toutes les applications en cours de fonctionnement s'arrêtent immédiatement dès lors que le master node tombe en panne car c'est ce dernier qui maintient toutes les informations sur l'état de l'ensemble des machine Spark incluant les applications qui y tournent. Ainsi, le master node est un point unique de défaillance pour un cluster Apache Spark utilisant son propre gestionnaire de ressources appelé "*Standalone Cluster Manager*".

Notons que Apache Spark supporte d'autres gestionnaires de ressources externes tels que "*Apache Mesos*" et "*Hadoop YARN*". Ces gestionnaires de ressources externes permettent de gérer simultanément un cluster Hadoop et un cluster spark installés sur les mêmes serveurs. Précisons au passage que Apache ZooKeeper est exploité en externe par *Apache Mesos* et en interne par *Hadoop YARN* pour résoudre le problème de présence d'un point unique de défaillance. Cependant, l'interface de programmation des applications (API) de Spark est indépendante du gestionnaire des ressources utilisé. Ainsi, une application Spark peut être déployée avec n'importe quel gestionnaire de ressources pris en charge par Spark sans que le code ne soit modifié et sans que les performances ne soient affectées.

Pour la mise en place d'un cluster Spark fonctionnant avec *Apache Mesos* ou *Hadoop YARN*, il est nécessaire d'avoir un minimun de connaissances sur ces technologies. Ce qui n'est pas le cas pour la mise en place d'un cluster Spark fonctionnant avec son propre *Standalone Cluster Manager* intégré. Le seul problème à utiliser Spark avec son propre gestionnaire de ressources est la présence d'un point unique de défaillance tel que mentionné dans le paragraphe précédent. Nous allons néanmoins adopter cette dernière installation en proposant une solution qui permet de résoudre ce problème et qui rend notre cluster Spark hautement disponible disponible.

Pour contourner le problème de présence d'un point unique de défaillance dans un cluster Spark utilisant son gestionnaire de ressources interne, la solution que nous proposons est d'utiliser plusieurs master nodes qui maintiendront les mêmes informations sur l'état du système. L'un de ces master nodes sera actif et les autres seront en attente. En cas de panne du master node actif, un des master nodes en attente est automatiquement élu comme actif et fait fonctionner le cluster sans aucune interruption des applications en cours. Dans un tel système, un cluster Apache ZooKeeper est utilisé pour coordonner le basculement entre les master nodes en cas de défaillance.

Dans ce document, nous allons décrire la procédure d'installation et de configuration d'un tel cluster Apache Spark hautement disponible et ne présentant pas de point unique de défaillance. Notons que nos serveurs fonctionnent sous CentOS 7 et que la version de Spark à installer est celle nommée `spark-2.3.0`.

2 Procédure d'installation et de configuration

2.1 Installation de Java

Comme pour la plupart des applications, Apache Spark nécessite une version récente de Java pour fonctionner. Pour installer la version récente OpenJDK de Java, nous pouvons exécuter la commande suivante :

```
# sudo yum install java -y
```

Nous pouvons vérifier la version installée de Java avec la commande suivante :

```
# sudo java -version
```

2.2 Installation de jps

On peut éventuellement installer `jps` sur toutes les machines de notre cluster pour voir l'identifiant des processus en cours de fonctionnement sur chacune de ces machines. La commande ci-dessous permet d'éffectuer cette installation :

```
# yum install java-1.8.0-openjdk-devel.x86_64
```

2.3 Ouverture des ports TCP

Si le pare-feu est activé dans les machines, il est nécessaire d'ouvrir et de persister quelques ports TCP spécifiques. On peut le faire avec les commandes ci-dessous :

```
# firewall-cmd --permanent --add-port=4040/tcp

# firewall-cmd --permanent --add-port=7077/tcp

# firewall-cmd --permanent --add-port=8080/tcp

# firewall-cmd --permanent --add-port=8081/tcp

# firewall-cmd --permanent --add-port=8880/tcp

# firewall-cmd --permanent --add-port=6660/tcp

# firewall-cmd --permanent --add-port=7770/tcp

# firewall-cmd --permanent --add-port=7337/tcp

# firewall-cmd --reload
```

On peut également vérifier les ports ouverts avec la commande suivante :

```
# firewall-cmd --list-ports
```

2.4 Installation de Zookeeper

Nous installons et configurons le cluster Zookeeper dans nos trois machines virtuelles nommées node2, node3 et node4. Pour cela, il suffit de suivre les instructions que nous avons fournies dans le document nommé "**Install ZooKeeper Cluster**" consacré à l'installation de ZooKeeper. Il est conseillé de ne pas exploiter les serveurs Zookepeer comme master nodes ou worker nodes du cluster Spark.

Pour le skockage des données issues de la coordination du cluster Spark par Zookeeper, il faut créer un dossier nommé par exemple "`spark`" dans le répertoire `ZK_HOME` de chaque server où zookeeper est installé. On peut le faire avec la commande ci-dessous à exécuter dans les node2, node3 et node4 hébergeant notre ensemble zookeeper :

```
# mkdir /usr/share/zookeeper-3.4.10/spark
```

2.5 Installation de Spark

Pour installer Spark dans notre cluster, il faut télécharger et décompresser le fichier `spark-2.3.0-bin-hadoop2.7.tgz` dans chacun des serveurs à utiliser comme master nodes et worker nodes. Nous pouvons par exemple éxécuter les commandes suivantes :

```
# cd /usr/share

# yum install wget

# wget https://archive.apache.org/dist/spark/
spark-2.3.0/spark-2.3.0-bin-hadoop2.7.tgz

# tar  -zxf spark-2.3.0-bin-hadoop2.7.tgz
```

Nous définissons les variables `SPARK_HOME` et `PATH` de façon permanente et globale pour tous les utilisateurs. Pour cela, nous éditons le fichier `/etc/profile` de chaque machine avec la commande suivante :

```
# vi /etc/profile
```

Ensuite, nous insérons les trois lignes ci-dessous juste avant la ligne contenant le texte `export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL`

```
export SPARK_HOME=/usr/share/spark-2.3.0-bin-hadoop2.7

export PATH=$PATH:/usr/share/spark-2.3.0-bin-hadoop2.7/bin

export PATH=$PATH:/usr/share/spark-2.3.0-bin-hadoop2.7/sbin
```

A la fin de cette modification, nous enregistrons et fermons ce fichier. Ensuite, nous éxécutons la commande ci-dessous pour que ces modifications soient immédiatement prisent en compte par le système :

```
# source /etc/profile
```

Si nous souhaitons libérer de l'espace disque à la fin de cette installation, nous pouvons supprimer le fichier `spark-2.3.0-bin-hadoop2.7.tgz` avec la commande ci-dessous :

```
# rm /usr/share/spark-2.3.0-bin-hadoop2.7.tgz
```

2.6 Configuration du cluster Spark

Pour configurer notre cluster Spark, nous devons créer un fichier nommé `spark-env.sh` et un fichier nommé `spark-defaults.conf` dans le répertoire `SPARK_HOME/conf` de tous les master et worker nodes de ce cluster. On peut créer et éditer ces fichiers avec les commandes ci-dessous :

```
# mkdir /tmp/spark-events
```

```
# cat > /usr/share/spark-2.3.0-bin-hadoop2.7/conf/spark-env.sh << "EOF"
SPARK_LOCAL_IP=10.10.10.X
SPARK_MASTER_HOST=10.10.10.X
SPARK_WORKER_PORT=8880
SPARK_WORKER_OPTS="-Dspark.worker.cleanup.enabled=true
-Dspark.worker.cleanup.appDataTtl=864000
-Dspark.worker.cleanup.interval=3600"
SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER
-Dspark.deploy.zookeeper.url=10.10.10.2:2181,10.10.10.3:2181,10.10.10.4:2181
-Dspark.deploy.zookeeper.dir=/usr/share/zookeeper-3.4.10/spark"
EOF

# cat > /usr/share/spark-2.3.0-bin-hadoop2.7/conf/spark-defaults.conf << "EOF"
spark.blockManager.port   6660
spark.driver.port   7770
spark.shuffle.service.port  7337
spark.driver.host   10.10.10.X
spark.eventLog.enabled    true
spark.eventLog.dir   file:///tmp/spark-events
spark.logConf    true
spark.driver.supervise    true
spark.dynamicAllocation.enabled   true
spark.shuffle.service.enabled    true
spark.local.dir    /tmp
spark.serializer    org.apache.spark.serializer.KryoSerializer
#spark.driver.cores   2
#spark.executor.cores   2
#spark.driver.memory   2g
#spark.executor.memory   4g
#spark.master    spark://10.10.10.5:7077,10.10.10.6:7077,10.10.10.7:7077
EOF
```

Conformément à l'attribution des adresses IP à nos machines virtuelles, nous devons remplacer dans le terme `10.10.10.X` la lettre `X` par le numéro du node concerné.

3 Démarrage et arrêt du cluster

3.1 Démarrage du cluster Zookeeper

On commence par démarrer chacune des trois machines du cluster Zookeeper :

```
# zkServer.sh start
```

Une fois que le cluster Zookeeper est en marche, on peut voir le status de chaque serveur :

```
# zkServer.sh status

# jps
```

3.2 Démarrage des master nodes

Après avoir mis en marche Zookeeper, on démarrage nos trois master nodes. Notons ici que nous avons choisi les machines nommées node5, node6 et node7 comme master nodes.

Le master nodex se lance avec la commande suivante :

```
# start-master.sh --ip 10.10.10.x
```

On peut utiliser la commande ci-dessous pour voir le status de chaque master node

```
# jps
```

3.3 Démarrage des worker nodes

Après la mise en route des master nodes, on démarre chacun des worker nodes. Chaque worker node se démarre avec la commande ci-dessous dans laquelle nos trois master nodes sont spécifiés :

```
# start-slave.sh spark://10.10.10.5:7077,10.10.10.6:7077,10.10.10.7:7077
```

On peut utiliser la commande ci-dessous pour voir le status de chaque worker node

```
# jps
```

3.4 Ouverture du shell Spark

Actuellement, le cluster est en marche et est parfaitement synchroniser avec le cluster ZooKeeper. On peut donc y éxécuter une application soit directement dans un terminal, soit interactivement via le shell Spark. Pour se connecter à ce cluster via le shell Spark, on peut éxécuter la commande suivante dans laquelle on spécifie les adresses IP et les ports d'écoute des master nodes :

```
# spark-shell --master spark://10.10.10.5:7077,10.10.10.6:7077,10.10.10.7:7077
```

On peut taper la commande ci-dessous pour quitter ce shell :

```
scala>:quit
```

3.5 Arrêt du cluster

Les master nodes s'arrêtent avec la commande suivante

```
# stop-master.sh
```

Les worker nodes s'arrêtent avec la commande suivante :

```
# stop-slave.sh
```

Chacun des serveurs Zookeeper s'arrête avec la commande suivante

```
# zkServer.sh stop
```

4 Conclusion

Nous avons montré dans ce document les étapes d'installation et de configuration d'un cluster Spark hautement disponible constitué des trois serveurs master node et d'un ensemble ZooKeeper à trois serveurs permettant de gérer le basculement entre les master en cas de défaillance d'un des master nodes. Cela nous sera absolument utile dans la suite pour transformer et structurer les données brutes avant de les stocker par exemple dans un cluster NoSQL Cassandra ou Solr. Par ailleurs, ce cluster Spark, nous sera également bénéfique pour la réalisation des aggrégations, des jointures et des traitement analytiques de nos différents datasets.

Références