UNIVERSITE GASTON BERGER

L'excellence au service du développement

INSTITUT POLYTECHNIQUE DE SAINT-LOUIS

Mini projet :

Apache HBase

Réalisé Par :

- **Daro Mbengue**

  Email : mbengue.daro@ugb.edu.sn

- **Michel Nassalang**

  Email : nassalang.michel@ugb.edu.sn

- **Ndeye Fatou Diouf Samba**

  Email : samba.ndeye-fatou-diouf@ugb.edu.sn

Encadreurs :

- Monsieur Mboup

Année Universitaire 2022 – 2023

Plan

Introduction

I. Description de la technologie HBase
II. Architecture
III. Principales fonctionnalités
IV. Avantages et Inconvénients
    1. Avantages
    2. Inconvénients
V. Installation et Déploiement
    1. Installation
    2. Déploiement
VI. Démo avec l'implémentation d'un exemple de use case

Conclusion

Introduction

Les applications Big Data nécessitent de stocker et de gérer des volumes massifs de données. Pour cela, il existe différentes technologies de stockage de données, dont Apache HBase. HBase est une base de données NoSQL distribuée et orientée colonne construite sur Apache Hadoop. Dans ce contexte, ce rapport va présenter la technologie HBase, son architecture, ses principales fonctionnalités, ses avantages et inconvénients, ainsi que l'installation et le déploiement de la technologie. Enfin, une démo sera effectuée avec l'implémentation d'un exemple de use case.

I. Description de la technologie HBase

HBase est un système de gestion de base de données non relationnelle orienté colonnes qui fonctionne sur HDFS (Hadoop Distributed File System). HBase
fournit un moyen tolérant aux pannes de stocker des ensembles de données éparses, ce qui est courant dans de nombreux cas d'utilisation Big Data.
Il est bien adapté au traitement des données en temps réel ou à l'accès aléatoire en lecture/écriture à de grands volumes de données.

Contrairement aux systèmes de bases de données relationnelles, HBase ne prend pas en charge les langages de requête structuré comme SQL ; en fait, HBase
n'est pas du tout un magasin de données relationnelles. Les applications HBase sont écrites en Java™ tout comme une application Apache MapReduce typique. HBase prend en charge l'écriture d'applications dans Apache Avro, REST et Thrift.

Un système HBase est conçu pour se mettre à l'échelle façon linéaire. Il comprend un ensemble de tables standard avec des lignes et des colonnes, comme une base de données traditionnelle. Chaque table doit avoir un élément défini comme une clé primaire, et toutes les tentatives d'accès aux tables HBase doivent utiliser cette clé primaire.

HBase s'appuie sur ZooKeeper pour la coordination haute performance. ZooKeeper est intégré à HBase, mais si vous exécutez un cluster de production, il est suggéré d'utiliser un cluster ZooKeeper dédié qui est intégré à votre cluster HBase.

HBase fonctionne bien avec Hive, un moteur de requête pour le traitement par lots des

données Big Data, pour permettre des applications de données volumineuses tolérantes aux pannes.

II. Architecture

L'architecture d'HBase est distribuée et orientée colonne. Elle est composée de plusieurs éléments :

- HMaster :

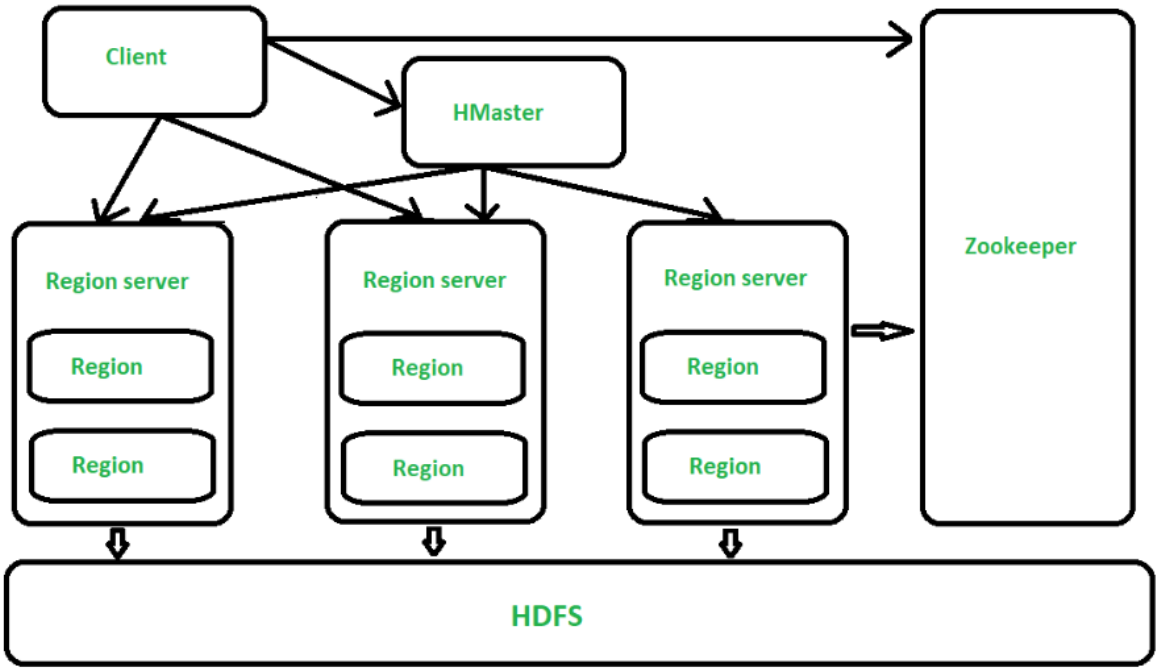
L'implémentation du serveur principal dans HBase est HMaster. C'est un processus dans lequel les régions sont attribuées à un serveur de région ainsi que les opérations DDL (création, suppression de table). Il surveille toutes les instances de serveur de région présentes dans le cluster. Dans un environnement distribué, le Master exécute plusieurs threads en arrière-plan. HMaster dispose de nombreuses fonctionnalités telles que le contrôle de l'équilibrage de charge, la reprise en cas de panne, etc.

- Serveur :

Les tables HBase sont divisées horizontalement par plage de clé de ligne en régions. Les régions sont les éléments de base de la construction du cluster HBase qui se composent de la distribution de tables et sont constituées de familles de colonnes. Le serveur de région s'exécute sur un DataNode HDFS qui est présent dans le cluster Hadoop. Les régions du serveur de région sont responsables de plusieurs choses, telles que la gestion, l'exécution, la lecture et l'écriture des opérations HBase sur cet ensemble de régions. La taille par défaut d'une région est de 256 Mo.

- Zookeeper :

Il est comme un coordinateur dans HBase. Il fournit des services tels que le maintien des informations de configuration, la dénomination, la fourniture de la synchronisation distribuée, la notification de défaillance du serveur, etc. Les clients communiquent avec les serveurs de région via Zookeeper.



III. Principales fonctionnalités

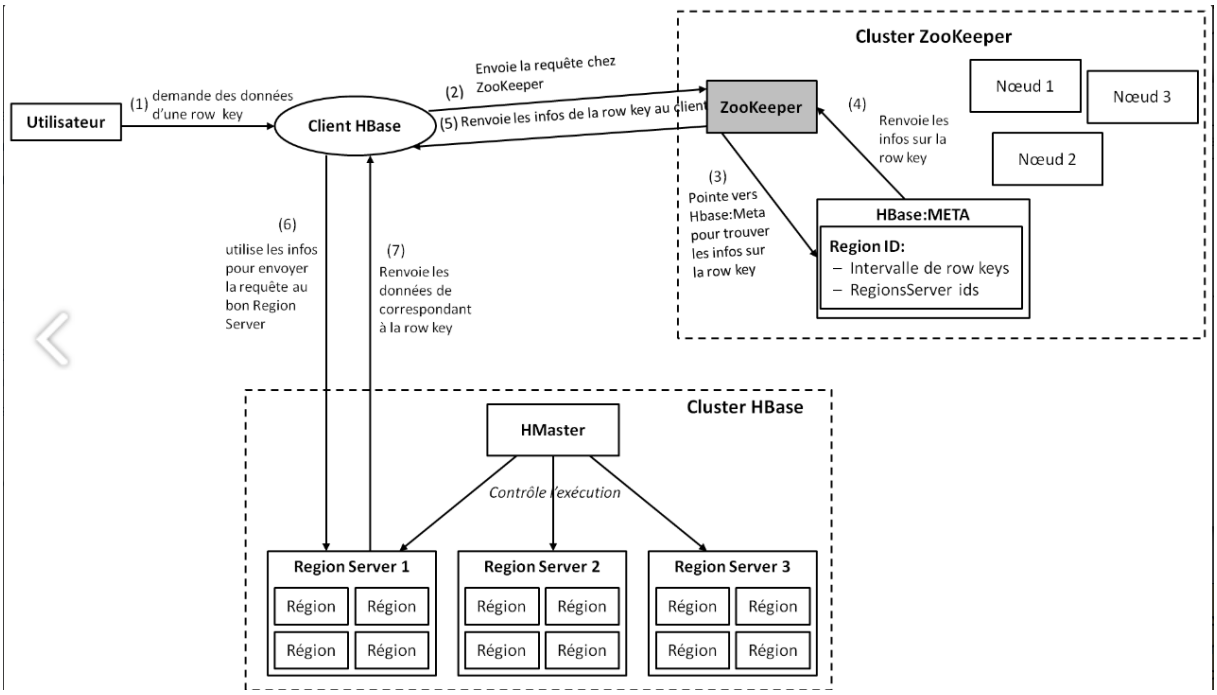
Apache HBase est une base de données NoSQL distribuée et orientée colonnes, qui est conçue pour stocker de grandes quantités de données non structurées sur un cluster de serveurs. Voici les principales fonctionnalités d'Apache HBase :

- Évolutivité horizontale : HBase peut gérer de grands volumes de données en ajoutant simplement plus de serveurs à un cluster. Il est conçu pour être distribué sur un grand nombre de nœuds, offrant ainsi une grande évolutivité horizontale.
- Haute disponibilité : HBase est conçu pour offrir une haute disponibilité, grâce à la réplication des données sur plusieurs nœuds. En cas de panne d'un nœud, les données peuvent être récupérées depuis d'autres nœuds.
- Compression de données : HBase prend en charge la compression de données pour réduire l'espace de stockage utilisé.
- Modèle de données orienté colonnes : HBase est conçu pour stocker des données non structurées dans un modèle de données orienté colonnes. Cela signifie que les données sont stockées dans des colonnes plutôt que dans des lignes, ce qui permet une récupération plus rapide des données.
- Transactions ACID : HBase prend en charge les transactions ACID (Atomicité, Cohérence, Isolation, Durabilité) pour garantir la cohérence des données et la fiabilité des opérations.
- Interface de ligne de commande : HBase fournit une interface de ligne de commande (HBase shell) pour interagir avec la base de données.
- API Java : HBase fournit une API Java pour les développeurs qui souhaitent interagir avec la base de données à partir de leurs applications.
- Intégration avec Hadoop : HBase est conçu pour s'intégrer facilement avec Hadoop, en utilisant HDFS (Hadoop Distributed File System) pour stocker les données et en

permettant l'exécution de traitements de données sur les données stockées dans HBase à l'aide de MapReduce.

Fonctionnement de HBase

Les régions sont distribuées sur le cluster de façon aléatoire et chaque nœud RegionsServer peut stocker une ou plusieurs régions.
Celle-ci sont répliquées entre les nœuds de façon à maintenir la disponibilité du cluster en cas de panne. Lors de l'ajout d'une ligne existante dans une
table (nouvelle version de la ligne), HBase retrouve la région contenant la valeur de la row key de la ligne et l'insère dans cette région.
Pour retrouver la région contenant la row key, HBase utilise une table de catalogue spéciale appelée "hbase:META". Cette table contient la liste des
RegionsServers disponibles, et la liste des intervalles de valeurs de row key pour chaque région de table. Elle est stockée dans un composant de
l'écosystème Hadoop appelé ZooKeeper, qui tourne sur un cluster différent du cluster sur lequel est installé Hbase. ZooKeeper est nécessaire parce
qu'à la différence d'Hadoop où la communication entre le client et le cluster se fait à l'intermédiaire du nœud de référence, dans HBase, le client
communique directement avec les nœuds RegionsServer sans passer par le HMaster. Le client n'a donc aucun moyen de connaître dans quel RegionsServer sont
situées les données dont il a besoin. Lorsqu'un client fait une requête sur une row key précise, ZooKeeper pointe vers la table hbase :
META pour récupérer les informations de la région contenant la row key, ensuite ZooKeeper renvoie cette information au client, qui va alors directement
s'adresser au Régions Serveur contenant la région. Finalement, la RegionsServer va traiter la requête et renvoyer au client les données de la row key.
La figure ci-après illustre le fonctionnement d'HBase lors d'une opération de lecture de données.



IV. Avantages et Inconvénients

1. Avantages

Avantages de HBase :

- Scalabilité horizontale : HBase peut être facilement étendue en ajoutant de nouveaux nœuds à un cluster Hadoop.
- Haute disponibilité : HBase est conçue pour fonctionner en continu, même en cas de défaillance d'un nœud ou d'un centre de données.
- Tolérance aux pannes : HBase est capable de récupérer automatiquement à partir d'une défaillance matérielle ou logicielle.
- Traitement de données massives : HBase peut gérer des quantités massives de données non structurées ou semi-structurées, ce qui la rend adaptée aux cas d'utilisation du Big Data.
- Hautes performances : HBase peut gérer de grandes quantités de données en temps réel avec des temps de réponse rapides.

2. Inconvénients

Inconvénients de HBase :

- Complexité : la configuration et la gestion d'un cluster HBase peut être complexe, en particulier pour les clusters de grande taille.
- Consistance : HBase offre une forte cohérence, ce qui peut entraîner des délais dans les mises à jour et des performances réduites.
- Pas adapté à toutes les tâches : HBase n'est pas la solution idéale pour toutes les tâches de stockage de données. Elle est particulièrement adaptée aux cas d'utilisation du Big Data, mais pas nécessairement aux opérations transactionnelles ou aux tâches de traitement de données simples.
- Développement limité : HBase est une technologie mature, mais son développement et son support peuvent être limités en comparaison à d'autres technologies NoSQL plus récentes.

En résumé, HBase est une base de données distribuée puissante qui offre des avantages significatifs pour le traitement de données massives. Cependant, sa complexité et son manque de flexibilité pour certaines tâches peuvent la rendre moins adaptée pour certains cas d'utilisation.

V. Installation et Déploiement

1. Installation

L'installation se fait à la suite du téléchargement hbase-3.0.0-alpha-3-bin.tar.gz ensuite on dépose ce fichier dans le dossier sharefolder. Une fois, la machine allumée par vagrant up et connectée par vagrant ssh. On retrouve le fichier, on le copie et le colle dans dans un dossier de la machine. Puis on le dézippe et on le place dans le dossier usr/local .

Ensuite, on entre dans le fichier .bashrc pour ajouter le HBASE_HOME. Pour nous permettre d'accéder à HBase.

Par la suite, on accède au fichier hbase-env.sh pour modifier la configuration de la java que doit utiliser HBase.

```
# Set environment variables here.

# This script sets variables multiple times over the course of starting an hbase process,
# so try to keep things idempotent unless you want to take an even deeper look
# into the startup scripts (bin/hbase, etc.)

# The java implementation to use.  Java 1.8+ required.
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.242.b08-0.el7_7.x86_64

# Extra Java CLASSPATH elements.  Optional.
# export HBASE_CLASSPATH=

# The maximum amount of heap to use. Default is left to JVM default.
# export HBASE_HEAPSIZE=1G

# Uncomment below if you intend to use off heap cache. For example, to allocate 8G of
# offheap, set the value to "8G". See http://hbase.apache.org/book.html#direct.memory
# in the refguide for guidance setting this config.
# export HBASE_OFFHEAPSIZE=1G

"usr/local/hbase/conf/hbase-env.sh" 178L, 10520C
```

2. Déploiement

L'utilisation de HBase se fait avec les commandes commencent par hbase suivi de l'action à faire. On peut entrer dans le shell avec hbase shell pour pouvoir faire des manipulations.

```
SHELL USAGE:
Quote all names in HBase Shell such as table and column names.  Commas delimit
command parameters.  Type <RETURN> after entering a command to run it.
Dictionaries of configuration used in the creation and alteration of tables are
Ruby Hashes. They look like this:

  {'key1' => 'value1', 'key2' => 'value2', ...}

and are opened and closed with curley-braces.  Key/values are delimited by the
'=>' character combination.  Usually keys are predefined constants such as
NAME, VERSIONS, COMPRESSION, etc.  Constants do not need to be quoted.  Type
'Object.constants' to see a (messy) list of all constants in the environment.

If you are using binary keys or values and need to enter them in the shell, use
double-quote'd hexadecimal representation. For example:

  hbase> get 't1', "key\x03\x3f\xcd"
  hbase> get 't1', "key\003\023\011"
  hbase> put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"

The HBase shell is the (J)Ruby IRB with the above HBase-specific commands added.
For more on the HBase Shell, see http://hbase.apache.org/book.html
hbase:012:0> quit
[vagrant@localhost /]$
```

```
Create a table with namespace=default and table qualifier=t1
  hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}
  hbase> # The above in shorthand would be the following:
  hbase> create 't1', 'f1', 'f2', 'f3'
  hbase> create 't1', {NAME => 'f1', VERSIONS => 1, TTL => 2592000, BLOCKCACHE => true}
  hbase> create 't1', {NAME => 'f1', CONFIGURATION => {'hbase.hstore.blockingStoreFiles' => '10'}}
  hbase> create 't1', {NAME => 'f1', IS_MOB => true, MOB_THRESHOLD => 1000000, MOB_COMPACT_PARTITION_POLICY => 'weekly'}

Table configuration options can be put at the end.
Examples:

  hbase> create 'ns1:t1', 'f1', SPLITS => ['10', '20', '30', '40']
  hbase> create 't1', 'f1', SPLITS => ['10', '20', '30', '40']
  hbase> create 't1', 'f1', SPLITS_FILE => 'splits.txt'
  hbase> create 't1', {NAME => 'f1', VERSIONS => 5}, METADATA => { 'mykey' => 'myvalue' }
  hbase> # Optionally pre-split the table into NUMREGIONS, using
  hbase> # SPLITALGO ("HexStringSplit", "UniformSplit" or classname)
  hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit'}
  hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit', REGION_REPLICATION => 2, CONFIGURATION => {'hbase.hregion.sca
 => 'true'}}
  hbase> create 't1', 'f1', {SPLIT_ENABLED => false, MERGE_ENABLED => false}
  hbase> create 't1', {NAME => 'f1', DFS_REPLICATION => 1}

You can also keep around a reference to the created table:

  hbase> t1 = create 't1', 'f1'

Which gives you a reference to the table named 't1', on which you can then
call methods.
hbase:010:0> create 'utilisateurs', {NAME => 'infos'}, {NAME => 'commandes'}
2023-05-06 00:07:45,410 INFO  [Registry-endpoints-refresh-end-points] client.RegistryEndpointsRefresher (RegistryEndpointsRefresher.java:ma
ints refresher loop exited.

ERROR: Failed contacting masters after 1 attempts.
Exceptions:
java.net.ConnectException: Call to address=10.0.2.15:16000 failed on connection exception: org.apache.hbase.thirdparty.io.netty.channel.Abs
Exception: Connection refused: /10.0.2.15:16000

For usage try 'help "create"'
```

VI. Démo avec l'implémentation d'un exemple de use case

J'ai rencontré des problèmes avec l'implémentation d'un exemple de use case pour la première fois car on avait installé la version client qui causait problème sans l'action du master.

La deuxième fois avec celle avec la configuration standalone, on avait commencé mais on ne comprend pas nos machines virtuelles ont arrêté de fonctionner brusquement et elles sont devenues inaccessibles.