**Apache HBase**

Table des matières

[Chapitre 1 INTRODUCTION 2](#_Toc101436080)

[Chapitre 2 : Modèle de données : 2](#_Toc101436081)

[Chapitre 3 : Modèle physique 3](#_Toc101436082)

[Chapitre 4 : Les opérations de base dans Hbase 5](#_Toc101436083)

[Chapitre 5 : Différence entre SGBDR & Hbase 5](#_Toc101436084)

[Chapitre 6 : Split dans Hbase 5](#_Toc101436085)

[Chapitre 7 : Hot-Spotting 6](#_Toc101436086)

[Chapitre 8 : Solutions pour Hot-spotting 6](#_Toc101436087)

[Chapitre 9 : Compaction (MAJOR & MINOR) 6](#_Toc101436088)

[Chapitre 10 : Bonnes pratiques 7](#_Toc101436089)

[Chapitre 11 : architecture (Ecriture & lecture dans Hbase) : 7](#_Toc101436090)

# 

# INTRODUCTION

Hbase est une base de données noSql orientée colonne, hautement scalable horizontalement (scale out) (ce qui permet de la requêter en temps réel = temps de réponse très faible)

1 table pour 1 use case métier, une table pour un référentiel (une case métier),

Exemple de référentiel : la liste de produits chez carrefour ça ne change pas tout le temps.

C’est une inspiration de BigTable de google GCP (cloud).

Basée sur HDFS

Pas de Sql, mais get, put delete, scan (pas d’update principe de hdfs)

# : Modèle de données :

D’une façon très simple une table Hbase est une map de map de map

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Row Key | |  |  |  |  | | --- | --- | --- | --- | | Column family | |  |  | | --- | --- | | Column Qualifier | Cell : version | | |

De point de vue architecture une table hbase est constituée obligatoirement de clé et column family et puis le reste de l’architecture est mis en place à l’écriture.

Les différentes informations dans une table hbase sont :

* Row key (clé de chaque ligne) ;
* Column family :
* Column Qualifier ;
* Cell
* Version

Cell est une combinaison de column qualifier, column family,

Les versions sont faites pour historiser en fonction du timestamp.

Pour des bonnes pratiques (afin de ne pas submerger le HDFS) il faut se contenter à créer 3 column family (3CF).

Les régions : sont créées au fur et à mesure de l’insertion qui auront lieu dans la table ; une région correspond à une plage de clés (de lignes) dans la table.

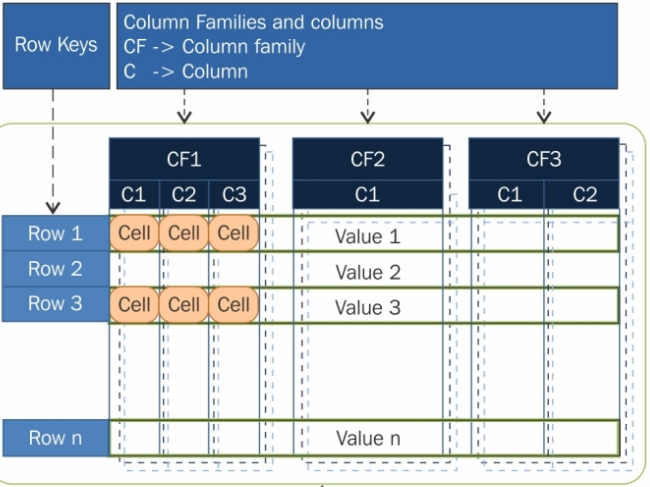
Region servers : ensemble des nœuds sur lesquels on a installé le service Hbase.

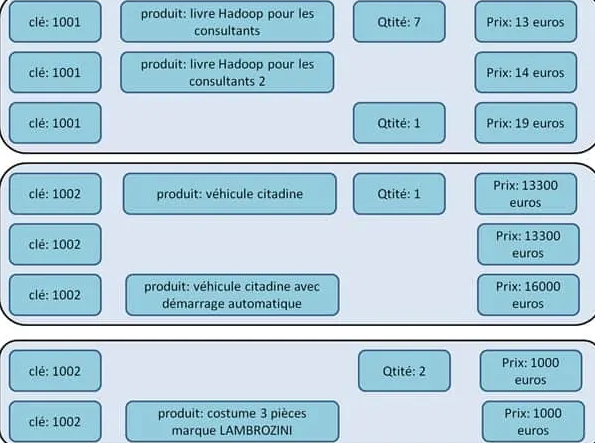
C’est avec l’activité que les régions sont créées (split région), c’est pour des raisons de performance.

# : Modèle physique

Le Buffer (le MemStor) qui devient un Hfile au moment de split. (Il existe un Memstore pour chaque column family)

Chaque Cell dans une table Hbase est une ligne dans le Hfile.





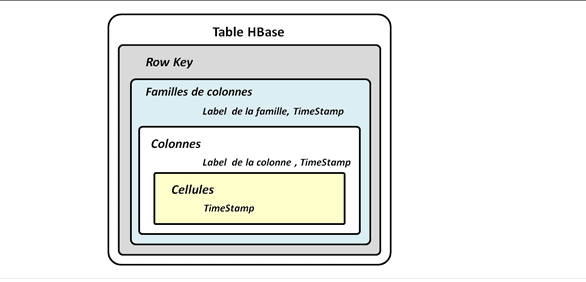


Figure 1: Imagination de table HBASE à trois dimensions(entonnoir)

# : Les opérations de base dans Hbase

Get (row key) : récupère une ligne

Put : insérer

Scan : récupère une plage de clé

Delete : supprimer une ligne (un terme très vaste car au moment qu’on va faire delete il ne va pas réellement supprimer la ligne mais il mettra un marqueur comme quoi la ligne n’existe pas (c’est au moment de la **MAJOR COMPACTION**) qu’il va réellement supprimer la ladite ligne).

# : Différence entre SGBDR & Hbase

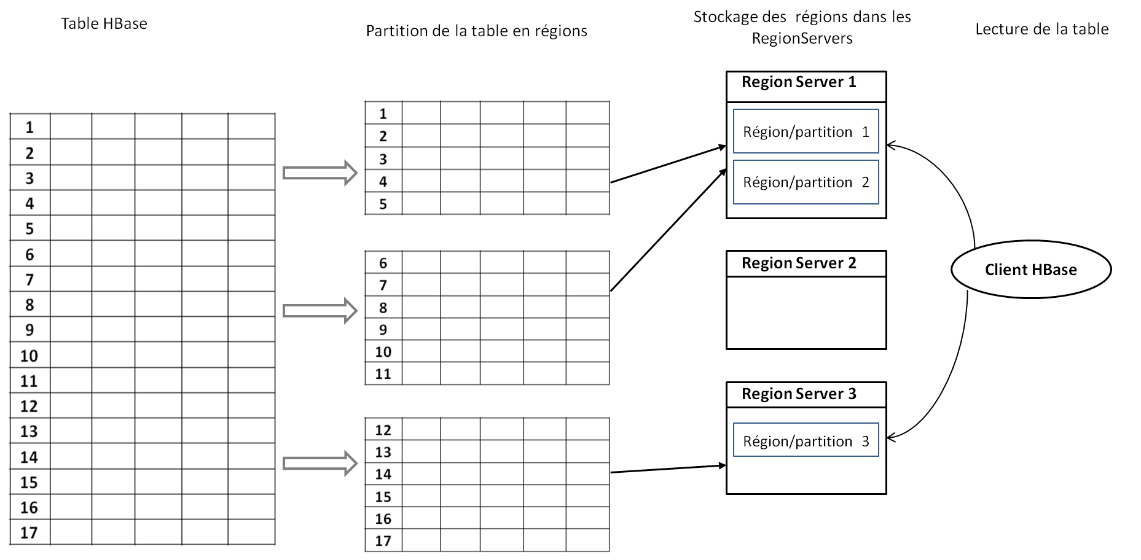
|  |  |  |
| --- | --- | --- |
|  | SGBDR | HBase |
| Modèle de données | Fixé | Flexible |
| Langage de requêtes | SQL | Get/put/scan/delete |
| Transactions | Multi-tables/multi-lignes | Une ligne uniquement |
| Jointure | Oui | Non (il faut job MapReduce) |
| Taile data | Térabyte | Milliers de pétabytes |
| Vitesse | 1req/seconde | Million req/seconde |
| Scalabilité | Après coup (scale up) | Conçue pour (scale-out) |
| Tolérance aux pannes | Maitre esclave (toujours dépendant) | Conçue pour |

# : Split dans Hbase

Le split est le phénomène qui subdivise une région (plage de clés) en deux régions filles dans un région server quand la region server atteint un certain seuil (taille).

Chaque région fille s’occupe de la moitié des clés de la région mère.

Au moment de split le Hmaster est responsable de décharger un region server (un datanode pour hdfs) d’une région (et déléguer la région à un autre region server dans le cluster hbase) afin de libérer ses ressources c’est ce qu’on appelle le **Load Balancing** (principe des calculs et de stockage distribué dans hadoop : en ordonnant à HDFS de nommer un datanode (region server) pour faire la tâche).



# : Hot-Spotting

Le hot-spotting est une erreur de conception de la table Hbase par faute de choisir une row key dont la valeur tout au début change rapidement (exemple de timestamp, ou Integer auto incrément) : il y aura donc un seule et unique region server qui s’occupera de tout les put, scan, get, le split etc …

Le region server sera donc saturé (ram, stockage, cpu) sachant qu’il existe beaucoup d’autres services hadoop qui tournent en même temps sur la même machine (spark, hive, hdfs, etc…).

En faisant ceci, on sort même du principe même du stockage et calculs distribués qui est : de distribuer chaque tache sur tous les nœuds du cluster.

# : Solutions pour Hot-spotting

La solution pour éviter le hot-spotting est le de réfléchir au moment de la création de la table à une rowkey qui qui change assez souvent à force de remplir une table.

**Quelques solutions** :

Une liste non exhaustive qu’on peut avoir comme solution on peut citer quelques-unes :

1. S’il s’agit par exemple d’un timestamp l’idéal est renverser le timestamp afin d’avoir des plages de valeurs très variés au furet à mesure des insertions donc la région mère se split et devient deux régions filles et que chacune s’occupe d’une certaine plage de valeur de la clé.
2. Mettre une rowkey un résultat d’une fonction de hachage de la clé préfixée par la clé elle même

# : Compaction (MAJOR & MINOR)

La compaction est le fait de réduire le nombre de Hfile.

**LA MINOR COMPACTION** :

Le but de cette compaction est surtout de faciliter le rôle de la major compaction ; elle est par défauts programmés à des intervalles de 24h dans hbase, consiste à réduire le nombre de de hfile selon un paramètre réglable. Si par exemple le facteur est de 3 ; on a 10 hfiles alors à la fin on obtient 3 hfiles résultant de la minor compaction. Sont importantes aussi pour optimiser les requêtes de scan (lecture sur disque)

**LA MAJOR COMPACTION** : C’est le but est d’avoir pour chaque région un seul et unique hfile.

Supprimer toutes les valeurs marquées lors des requêtes de delete. Il génère beaucoup de flux réseaux et bouffe beaucoup de ressources en cpu et en ram d’où la nécessité de la programmer à des moments où la table est moins requêtée (la nuit par exemple parce si la table n’est pas utilisée la nuit)

# : Bonnes pratiques

Avoir un schéma avant la création de la table qui répond au besoin métier. (Répond à toutes les requêtes).

Ne pas mettre plus de 3 CF par table car ça explose les ressources de la table.

Dénormalisation & éclatement des données car pas de jointures (sauf avec map reduce mais compliqué)

Eviter le TTL dont le principe est de régler hbase à supprimer les lignes chaque un laps de temps.

# : architecture (Ecriture & lecture dans Hbase) :

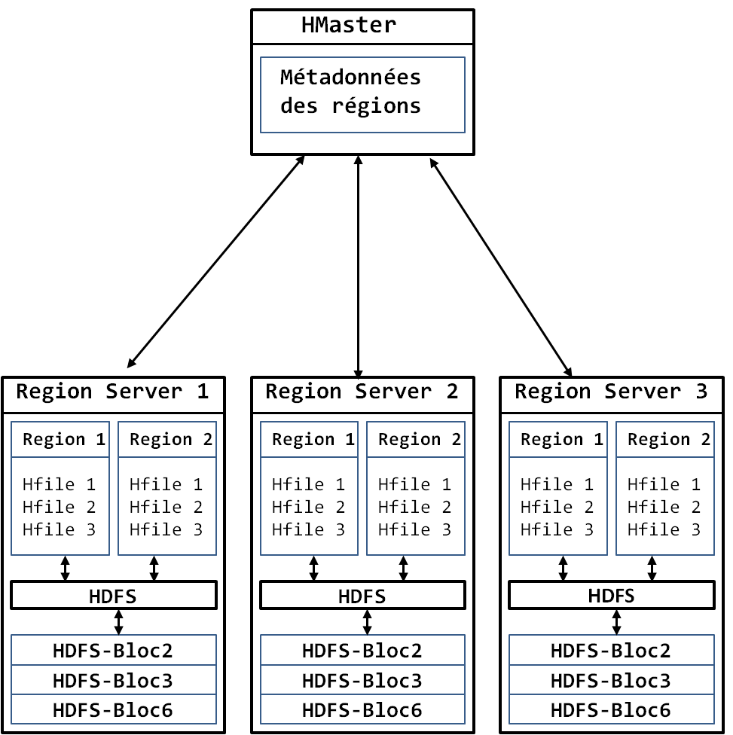
HBase est un SGBD distribué. Comme Hadoop, HBase s'installe sur un cluster en architecture. Le stockage des données est distribué sur les Regions Servers qui sont gérés par le HMaster. Le HMaster gère les métadonnées des tables HBase et coordonne l'exécution des activités des Regions Servers, tandis que les Regions Servers effectuent les opérations de lecture/écriture de données dans le cluster. La gestion du volume de données se fait par l'ajout des Regions Servers supplémentaires dans le cluster.

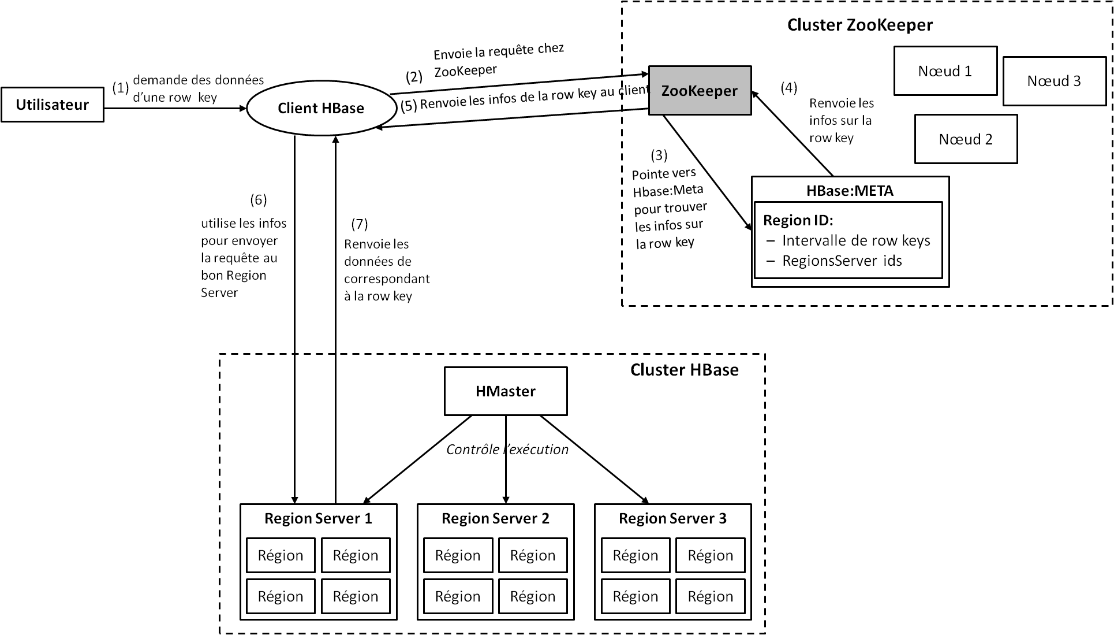
**ZooKeepe**r est nécessaire parce qu'à la différence d'Hadoop où la communication entre le client et le cluster se fait à l'intermédiaire du nœud de référence, dans HBase, le client communique directement avec les nœuds Regions Server sans passer par le HMaster. Le client n'a donc aucun moyen de connaître dans quel Regions Server sont situées les données dont il a besoin

***ECRITURE :***

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 ***ZooKeeper***, (chaque mem store responsable d’une colonne family). Il faut noter quand on écrit on écrit d’abord dans le Wal (write Head Log : à chaque fois qu’on envoie une requête de modification de la table le wal l’enregistre et grâce à lui on peut reconstruire toute la vie d’une table)

Le sens de l’écriture : dans le wal et le memstore en même temps.





***LECTURE :***

De la même façon que l’écriture :

 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 Regions Server contenant la région. Finalement, la Regions Server 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.

Le sens de la lecture :

1. Vérification d’abord dans le bloc cached (le bloc cached c’est une mémoire vive qui sert de mémoire et enregistre les requêtes qui ont été adressées à la table hbase.)
2. Vérifie dans le memstore (buffer)
3. Vérifie et scan les hfiles de la table.