

Apache HBase : un SGBD NoSQL orienté colonne

Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

DataCloud – Master 2 SAR 2020/2021



Le stockage de données dans l'Eco-système Hadoop

HDFS :

- ✓ stockage de grand volume de données
- ✗ Accès séquentiel aux données :
 - ⇒ accès à une donnée ponctuelle stockée dans un fichier = un scan complet du cluster
 - ⇒ **coûteux en temps et en calcul**

Comment indexer et structurer des grandes masses de données pour un accès direct (random access) ?

Utilisation d'un SGBD



Bref historique

- Nov 2006 : Google publie un papier sur BigTable
- Oct 2007 : Premier Hbase "utilisable"
- Mai 2010 : HBase devient un projet Apache top-level
- Fev 2015 : Hbase 1.0.0
- Sep 2020 : HBase 2.2.6

Caractéristiques principales de HBase

Système d'indexation distribué

⇒ répartition de charge

Repose sur un système de fichiers distribué fiable (HDFS par défaut)

⇒ tolérance aux pannes, robustesse des données

Écritures/lectures directes sur un très grands ensemble de données

⇒ accès accélérés

Stockage NoSQL :

- orienté colonne :

⇒ adapté aux traitements analytiques en ligne (OLAP)

- schéma clé → valeur

⇒ accès rapide à une valeur par sa clé

- triée

⇒ permet de récupérer les valeurs par intervalle de clés

Caractéristiques principales de HBase

Passage à l'échelle horizontal et linéaire

⇒ nombre de machines $\times 2$ = stockage et puissance de calcul $\times 2$

Partitionnement automatique des données en régions

⇒ distribution et répartition des données transparentes pour l'utilisateur

Basculement automatique en cas de serveur de données défaillant

⇒ tolérance aux pannes transparentes pour l'utilisateur

Accès via Map-Reduce

⇒ traitement massivement parallèles

API Java

⇒ intégration naturelle dans l'écosystème Hadoop

Namespace1

Table1	ColumnFamily1				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
							vers2	val11		
	vers1	val2					vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

Namespaces

Namespace1

Table1	ColumnFamily1				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
							vers2	val11		
	vers1	val2					vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

Definition

- Un groupement logique de table

Caractéristiques

- Permet d'isoler des tables pour des raisons de quotas, de restrictions géographiques, de sécurité
- Deux namespace existent déjà par défaut
 - **hbase** : Contient toutes les tables des méta-données de HBase
 - **default** : namespace par défaut lorsque aucun namespace n'est spécifié à la création d'une table

Table

Namespace1										
Table1	ColumnFamily1				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
							vers2	val11		
	vers1	val2					vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

Définition

- Élément servant à organiser les données dans HBase
- Le nom d'une table est une chaîne de caractères
- Désignée de manière non ambiguë en préfixant son nom par le nom de son namespace séparé par ' : '

nom_namespace:nom_table

Namespace1										
Table1	ColumnFamily1				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
							vers2	val11		
	vers1	val2					vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

Définition

- Permet d'organiser les données dans une table
- Une ligne est identifiée par une clé unique : **RowKey**
- La Rowkey n'a pas de type, c'est un tableau d'octets.

Namespace1										
Table1	ColumnFamily1				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
							vers2	val11		
	vers1	val2					vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

Définition

- Regroupe les données au sein d'une ligne
- Toutes les lignes de la table ont les mêmes ColumnFamily, pouvant être peuplée ou pas
- Au moins une à la création de la table dans HBase

Namespace1										
Table1	ColumnFamily1				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
							vers2	val11		
	vers1	val2					vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

Définition

- Permet de subdiviser les columnfamily
- Désignée par une chaîne de caractères appelée **column qualifier**
- Spécifiée au moment de l'insertion de la donnée
- Non typée, le nom est un tableau d'octets

Namespace1										
Table1	ColumnFamily				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
							vers2	val11		
	vers1	val2					vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

Définition

- Identifiée par la combinaison d'un RowKey, de la Column Family et de la Column
- Les données stockées dans une cellule sont les valeurs de la cellule
- On peut stocker différentes versions de la cellule (ou timestamp)

Namespace1										
Table1	ColumnFamily1				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
	vers1	val2					vers2	val11		
							vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

Définition

- Les valeurs au sein d'une cellule sont versionnées
- Les versions sont identifiées par défaut par un timestamp (de type long)
- Le nombre de versions stockables par cellule est paramétrable

Namespace1										
Table1	ColumnFamily1				ColumnFamily2					
	Col1		Col2		Col3		Col4		Col5	
Rowkey1	vers2	val1	vers1	val6	vers1	val7	vers3	val10		
							vers2	val11		
	vers1	val2					vers1	val12		
Rowkey2	vers1	val3					vers1	val13	vers1	val16
Rowkey3			vers1	val5	vers1	val9	vers1	val14	vers1	val15

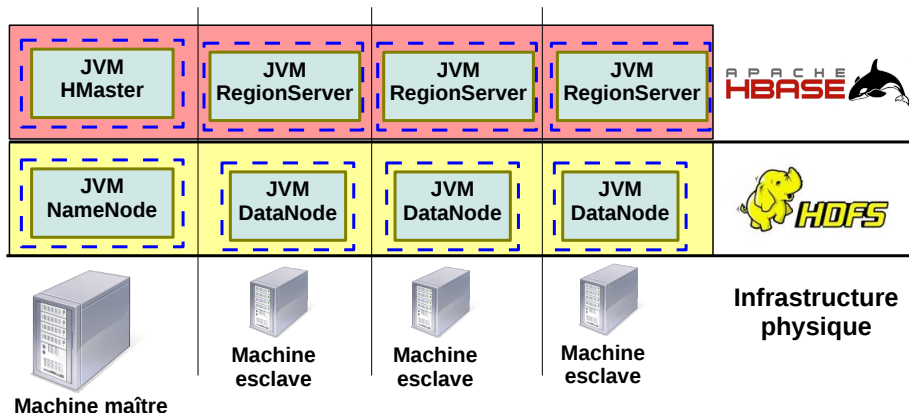
Définition

- Une valeur est une donnée atomique de la base
- Non typée et stockée au format binaire
- Les valeurs null ne sont pas matérialisées (aucun stockage nécessaire)
- Désignée par une clé multi-dimensionnelle :
(rowkey, column family, column, version)

Désignation complète d'une valeur dans HBase

namespace :table(rowkey, column family, column, version) → val

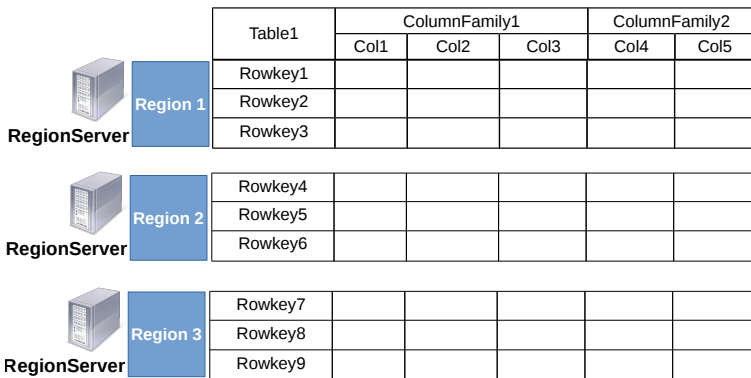
Architecture globale



Une architecture maître esclave

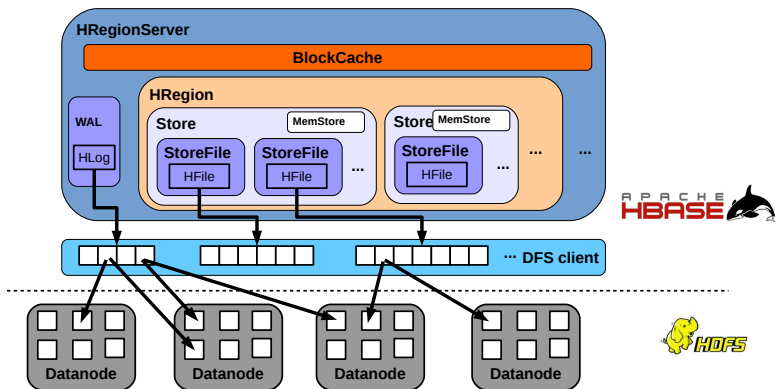
- Nœud maître = Hmaster, Nœud esclave = RegionServer
- Correspondance directe avec le cluster HDFS

Découpage des données sur le cluster



Région

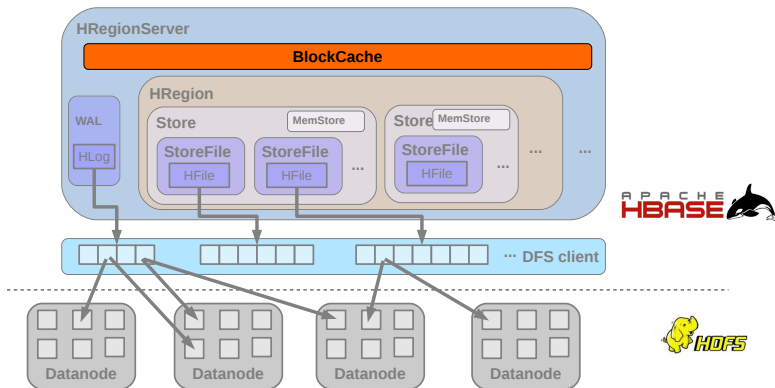
- Sous ensemble contiguë de lignes de la table
- Stockée sur un nœud physique esclave et triée selon la rowkey
- Identifiée par un rowkey minimum et un rowkey maximum



Caractéristiques principales

- Point d'entrée pour accéder à une donnée
- Propose les services :
 - Données (get, put, delete, next, etc.)
 - Region (splitRegion, compactRegion, etc.)

RegionServer : BlockCache

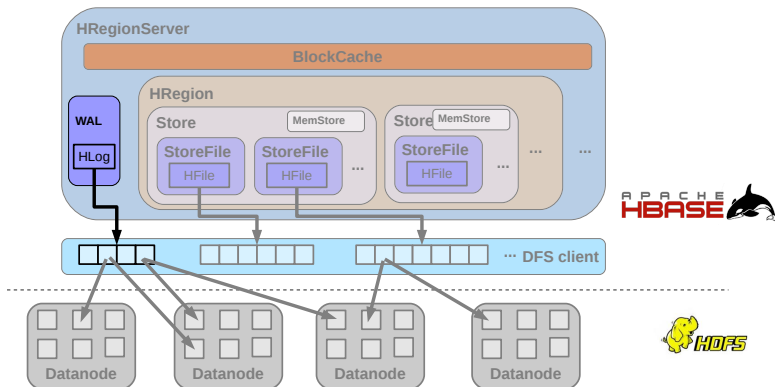


Caractéristique du BlockCache

Cache LRU activé par défaut pour toutes les tables

⇒ Toute opération de lecture est chargée dans le cache LRU.

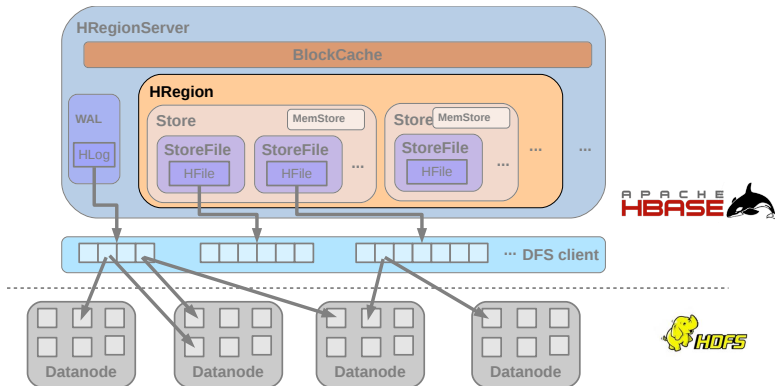
RegionServer : Write Ahead Log (WAL)



Caractéristiques des WAL

- Loguent les ajouts/mises à jour fait sur le RegionServer
- Garantissent la durabilité de la donnée en cas de défaillance
- Stockent les informations dans un fichier HDFS HLog

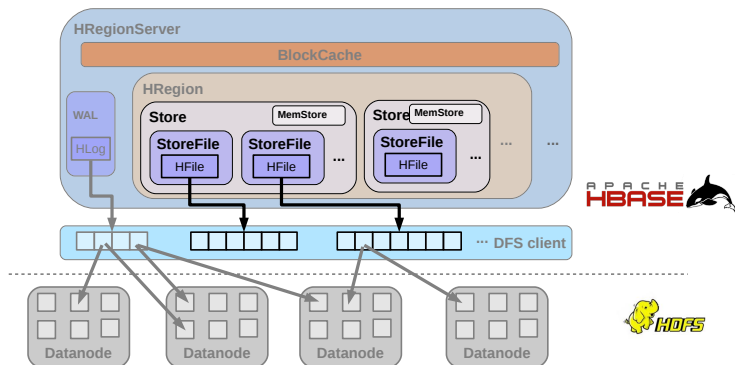
RegionServer : HRegion



Caractéristiques des HRegion

- Représente une région
- Gère un sous ensemble de lignes contiguës d'une table Hbase

RegionServer : Store



Caractéristiques des Stores

- Représente une columnFamily de la région
- Appartient à une seule région
- Le cache memStore stocke toutes les écritures relatives à la partition
- Stocke les données physiquement dans plusieurs fichiers HDFS.

Résumé

- Une table est segmentée en plusieurs partitions (= région)
- Une région d'une table est gérée par un RegionServer
- Un RegionServer gère plusieurs Regions
- Une Region contient plusieurs Stores
- Chaque Store gère une ColumnFamily d'une table
- Un Store gère un MemStore et plusieurs StoreFile
- Un StoreFile gère un fichier de stockage de la partition

Caractéristiques

- Coordonne et surveille les RegionServer : les remplace si besoin
- Interface pour tout changement des meta-données du système (table hbase :meta) :
ex : création/suppression/modif d'une table ou d'une col family
- Assure l'équilibre de charge entre RegionServer :
(dé)assigne/déplace les régions
- Peut être répliqué

En cas de panne :

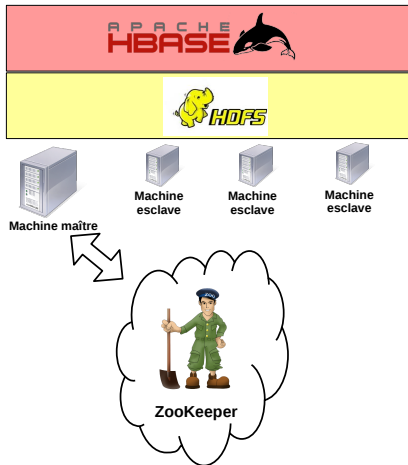
- un des réplicas prend sa place
- pendant la gestion de la panne, HBase peut continuer à fonctionner car les clients s'adressent directement aux RegionServer

Caractéristiques

- Logiciel Apache de de coordination de systèmes distribués
- Entrepôt de méta-données partagées
- Surveillance et notification d'événements

Rôle pour HBase

- Notifie le HMaster en cas de panne d'un RegionServer
- Stocke la localisation de la table hbase :meta dans le cluster



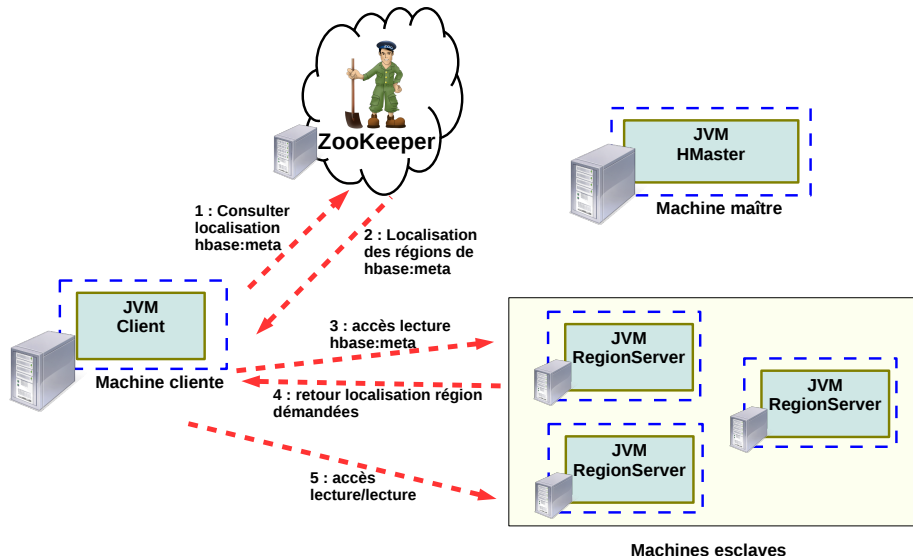
Caractéristiques

- Maintien la liste de toutes les régions du système
- La localisation des régions de cette table est stockée dans ZooKeeper

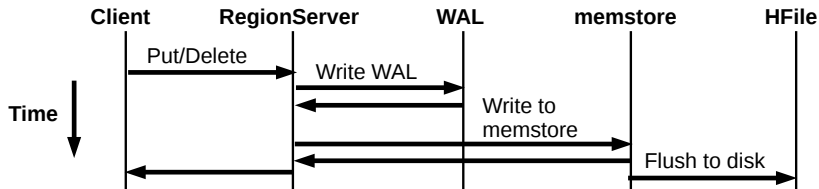
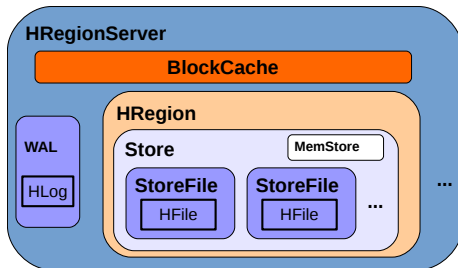
hbase

meta	info		
	regioninfo	server	serverstartcode
(nom_table, clé de départ, id_region)	<info diverse>	<adresseIP:port du serveur>	<date a laquelle la region a été crée sur le serveur>

Accès à une donnée depuis un client



Cas d'une écriture



les API

- un shell dédié en lançant la commande `hbase shell`
- API JAVA native
- API externes :
 - REST
 - Thrift

Commandes générales

- **status** : retourne l'état des RegionServers du système
- **version** : retourne la version courante de Hbase sur le système
- **help** : affiche une aide sur l'utilisation du shell HBase ainsi que la liste des commandes possibles
- **table_help** : affiche une aide sur l'utilisation des commandes manipulant les tables du système
- **whoami** : affiche des information sur l'utilisateur courant

Commandes pour les namespace

- **Créer un namespace**

```
create_namespace 'my_ns'
```

- **Effacer un namespace** : le namespace ne doit pas contenir de table

```
drop_namespace 'my_ns'
```

- **Liste des namespace dans le système**

```
list_namespace
```

- **Liste des tables présentes dans un namespace**

```
list_namespace_tables 'my_ns'
```

- **Modifier les attributs d'un namespace**

```
alter_namespace 'my_ns', {METHOD=>'set', 'PROPERTY_NAME'=>'PROPERTY_VALUE'}
```

- **Lire les attributs d'un namespace**

```
describe_namespace 'my_ns'
```

Commandes table

- **create** : créer une table
`create '<table name>', '<colfam1>', '<colfam2>', ..., '<colfamN>'`
- **list** : Lister les tables dans HBase.
- **disable** et **enable** : désactiver/activer une table. Désactivation obligatoire avant suppression
- **is_disabled** et **is_enabled** : tester la désactivation/l'activation
- **describe** :Afficher les attributs d'une table
- **scan** : Afficher tout le contenu d'une table
- **drop** : effacer une table de Hbase. la table doit être désactivée.
- **exists** : tester si une table existe
- **count** : affiche le nombre de ligne dans la table
- **truncate** : désactive + efface + recrée la table

Modifier les attributs d'une table.

- Commande alter
- Exemple :
 - interdire l'écriture sur une table
`alter 't1', READONLY`
 - effacer une columnFamily
`alter '<table name>', 'delete' => '<column family>'`

Manipuler des données dans une table

- **Ecrire/mettre à jour une donnée :**

```
put '<table name>','<rowid>','<colfamily:colname>','<value>'
```

- **Lire une ligne d'une table :**

```
get '<table name>', '<rowid>'
```

- **Lire une colonne spécifique d'une ligne :**

```
get '<tablename>', '<rowid>', {COLUMN => '<colfam>:<colname>'}
```

- **Effacer une cellule :**

```
delete '<tablename>', '<row>', '<colname>', '<timestamp>'
```

API Java : Connexion/déconnexion à Hbase

Connexion à Hbase

```
Configuration conf = HBaseConfiguration.create();
conf.set("hbase.zookeeper.quorum", "server1.com,server2.fr");
Connection c = ConnectionFactory.createConnection(conf);
...//code client Hbase
c.close();//fermeture connexion
```

Connexion/déconnexion à une table

```
Connection c = ..;
TableName tableName = TableName.valueOf("ma_table");
Table table = c.getTable(tableName);
...
table.close();//fermeture desc table
```

Caractéristiques

- "hbase.zookeeper.quorum" = machines serveurs de ZK (par def : localhost)
- **Recommandation** : une unique connexion par JVM cliente
 - une connexion à ZooKeeper est coûteuse
 - les données de ZK sont en cache sur la machine cliente

RAPPEL

Dans Hbase les données ne sont pas typées et sont stockées au format binaire :

⇒ Nécessité de convertir toutes les données du programme :

- du type java d'origine vers `byte[]` avant une écriture
- de `byte[]` vers le type Java d'origine après une lecture

La classe utilitaire Bytes

Vers le binaire

```
static byte[] toBytes(String s);
static byte[] toBytes(boolean b);
static byte[] toBytes(long val);
static byte[] toBytes(float f);
static byte[] toBytes(int val);
...
```

Depuis le binaire

```
static String toString(byte[] bytes);
static boolean toBoolean(byte[] bytes);
static long toLong(byte[] bytes);
static float toFloat(byte[] bytes);
static int toInt(byte[] bytes);
...
```

API Java : création/suppression d'une table

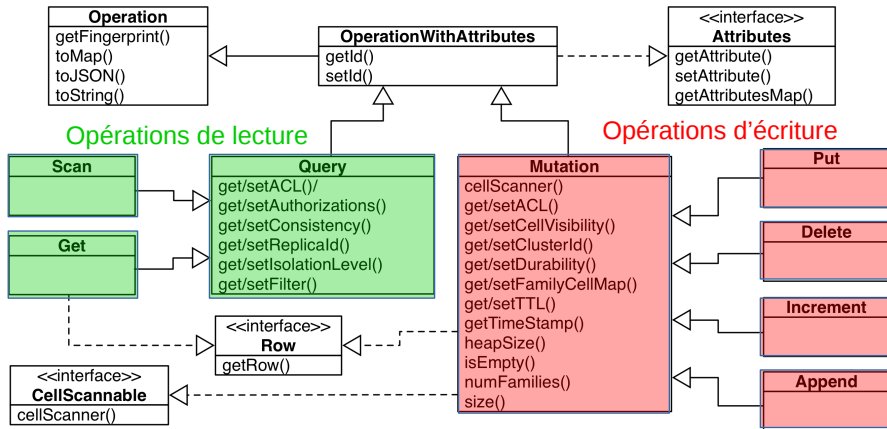
Création d'une table

```
Connection c = .....;
Admin admin = c.getAdmin();
HTableDescriptor desc = new HTableDescriptor(
    TableName.valueOf(namespace+"."+nametable));
desc.addFamily(new HColumnDescriptor("col_fam1"));
desc.addFamily(new HColumnDescriptor("col_fam2"));
...// add other col families
admin.createTable(desc);
```

Suppression d'une table

```
Connection c = .....;
Admin admin = c.getAdmin();
HTableDescriptor desc = new HTableDescriptor(
    TableName.valueOf(namespace+"."+nametable));
admin.disableTable(desc.getTableName());
admin.deleteTable(desc.getTableName());
```

API Java : Opération de lecture/écriture sur les tables



Caractéristiques

- Objet caractérisant une ou plusieurs écritures/ modifications d'une ligne
- constructeurs : `Put(byte[] rowkey)` OU `Put(byte[] rowkey, long def_ts)`
- Ajouter une colonne à créer/à modifier :

```
Put addColumn(byte[] family, byte[] col, byte[] val);  
Put addColumn(byte[] family, byte[] col, long ts, byte[] val);
```

Exemple d'utilisation

```
Table table = ....; //connexion à une table  
Put put = new Put(Bytes.toBytes("row1"));  
//ajout d'une valeur v1 dans la cellule <row1,cf1:c1>  
put.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("c1"), Bytes.toBytes("v1"));  
//ajout d'une valeur v2 dans la cellule <row1,cf1:c2>  
put.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("c2"), Bytes.toBytes("v2"));  
  
table.put(put); //envoi de la requête sur la table
```

les Deletes

- Objet caractérisant la suppression partielle ou totale des colonnes d'une ligne
- constructeur : `Delete(byte[] rowkey)`

les Appends

- Objet caractérisant une opération atomique de read-modify-write sur la cellule d'une ligne
- constructeur : `Append(byte[] row)`
- méthode : `Append add(byte[] family, byte[] col, byte[] value)`
⇒ création d'une nouvelle version de la cellule en concaténant l'ancienne valeur avec la nouvelle valeur

Caractéristiques

- Objet caractérisant la lecture partielle ou totale d'une ligne
- constructeur : `Get(byte[] rowkey)`
- Ajouter des critères de sélections :
`Get addFamily(byte[] family);`
`Get addColumn(byte[] family, byte[] qualifier);`
`Get setTimeRange(long minStamp, long maxStamp);`
`Get setMaxVersions(int maxVersions);`
- le résultat d'une lecture est un objet de type `Result` qui contient toutes les cellules qui correspondent aux critères de la requête

Exemple d'utilisation

```
Table table = ....; // connexion à une table
Get get = new Get(Bytes.toBytes("row1"));
get.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("c1"));

Result res = table.get(get); // envoi de la requête sur la table
byte[] val = res.getValue(Bytes.toBytes("cf1"), Bytes.toBytes("c1"));
System.out.println(Bytes.toString(val));
```

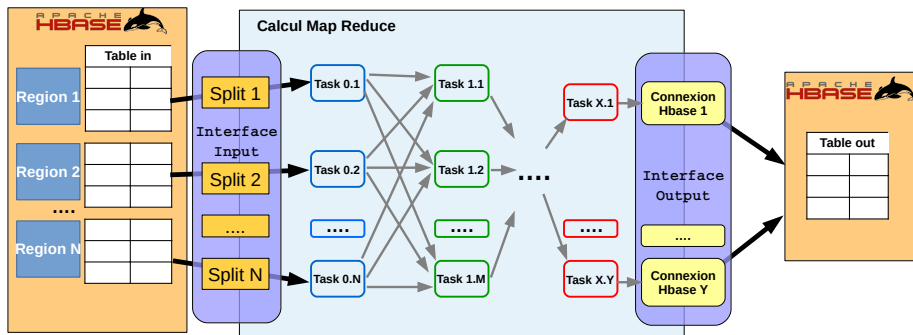

Caractéristiques

- Objet caractérisant la lecture séquentielle de plusieurs lignes
- constructeurs :
 - `Scan()` : toute la table
 - `Scan(byte[] startRowKey)` : à partir d'une ligne donnée
 - `Scan(byte[] startRowkey, byte[] stopRowKey)` : sur une portion
- mêmes méthodes que `Get`

Exemple d'utilisation

```
Table table = ....; // connexion à une table
Scan scan = new Scan(Bytes.toBytes("rowX"), Bytes.toBytes("rowY"));
scan.addColumn(Bytes.toBytes("cfz"), Bytes.toBytes("col"));
ResultScanner results = table.getScanner(scan);
for(Result res: results){
    System.out.println(res);
}
```

Hbase et Job MapReduce



Principes

- Une tâche de début est affecté à une région
 - Une tâche de fin écrit dans la base
- ⇒ **Toute tâche de début ou de fin est un client Hbase**

Atomicité

- Toute mutation est atomique pour une ligne entière et peut être :
 - soit "success" \Rightarrow réussite complète
 - soit "failed" \Rightarrow échec complet
- L'ordre de mutations concurrentes pour une ligne se fait sans entrelacement.
ex : si "a=1,b=1" || "a=2,b=2" alors soit "a=1,b=1" ou soit "a=2,b=2"
- L'atomicité n'est pas garantie sur plusieurs lignes

Cohérence

- Tout get sur une ligne complète retournera une version de la ligne qui a existé dans l'histoire de la table :
 - si 1 *get* || plusieurs mutations, alors le *get* retournera une ligne complète qui a existé à un point donné dans le temps entre les mutations
- Un scan n'est pas une vue cohérente de la table
- Toute ligne retournée par un scan est cohérente et est au moins aussi récente que le début du scan

Durabilité

- Toute donnée visible est durable \Rightarrow un read concerne forcément une donnée validée
- Toute opération acquittée réussie est durable
- Toute opération acquittée échec ne sera pas durable

[1] <http://hbase.apache.org/book.html>

[2] <https://www.tutorialspoint.com/hbase>

[3] HBase : The Definitive Guide, 2nd Edition, Lars George, O'Reilly Media, Inc., ISBN : 9781491905845