

# Apache HBase : un SGBD pour le BigData

Jonathan Lejeune

UPMC/LIP6-INRIA

CODEL – Master 2 SAR 2017/2018



## 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

## Les plus populaires

Oracle Database, Microsoft SQL Server, MySQL, IBM DB2

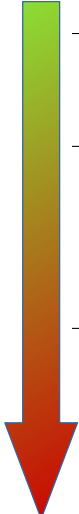
## Principaux avantages

- Données structurées, modélisation intuitive des données
- Normalisation des données (3NF, 3BCNF, ...)
- Indexation des données, Optimisation d'accès
- Un langage d'interrogation : le SQL
- Simple à administrer/à déployer car centralisé sur un serveur
- Propriétés ACID :
  - **A**tomacité : une transaction se fait au complet ou pas du tout
  - **C**ohérence : l'état du système reste valide à chaque transaction
  - **I**solation : Aucune dépendance possible entre les transactions
  - **D**urabilité : une transaction confirmée demeure enregistrée

# Les limites des SGBD relationnels

Hypothèse : nb lectures  $\gg$  nb écritures

Nb Users	Problème	Solution possible
1000	tout va bien	
10000	serveur central du SGBD de + en + chargé en I/O et CPU	ajouter machines esclaves + load balancer : lectures pour les esclaves, écritures pour le maître
100000	Les esclaves sont de + en + chargés	ajouter un service de cache (Memcached, Redis, ...) perte des garanties de cohérence entre cache et database
>100000	les écritures font goulot d'étranglement	remplacer le master par une machine plus puissante (=vertical scaling) augmentation des coût d'exploitation



## Limites

- Inadapté à la vélocité des données
- Les traitements classiques passent difficilement à l'échelle :
  - CRUD : Create, Read, Update, Delete sont de + en + lent
  - Jointures doivent maintenir les relations entre les données
- Les index sont difficiles à maintenir

**Besoin de d'améliorer/modifier le modèle de représentation des données**

## Les plus populaires

HBase, Cassandra, Dynamo, MongoDB, Google BigTable, ....

## Caractéristiques communes

- Les APIs d'interrogation sont simplifiées (pas de SQL)
- Le schéma des données est très flexibles
- Pas de relation complexe entre les données (pas de join)
- Les données et les calculs sont distribuées

# Les types de SGBD NoSQL

## Clé-valeur

- Une table de hachage distribuée : une clé  $\rightarrow$  une donnée

## Orienté colonne

- Les clés désignent plusieurs colonnes qui sont regroupées en famille.

## Orienté document

- Basé sur des documents semi-structurés et respectant un format (JSON)

## À base de graphe

- Les tables sont représentées par un ensemble de nœuds (un nœud = un tuple)
- Un arc existe entre deux tuples, si ils ont un lien relationnel (ex : un pneu avec une voiture)

# Résumé : NoSQL vs. SGBD relationnel

	<b>NoSQL</b>	<b>SGBD relationnel</b>
<b>Structure des données</b>	sans schéma, colonnes non fixées	structure fixée sur l'ensemble des tables
<b>Taille</b>	très grandes tables (milliard de lignes)	taille moyenne (million de lignes)
<b>Traitement</b>	analytique et non transactionnel	transactionnel
<b>Stockage</b>	généralement en colonne	généralement en ligne
<b>Passage à l'échelle</b>	linéaire et horizontal	vertical sur un serveur central
<b>Requetage</b>	simple (put/get)	élaboré (SQL, PL/SQL)
<b>valeur nulle</b>	non matérialisée	stockée



# Exemple illustratif : réducteur d'URL

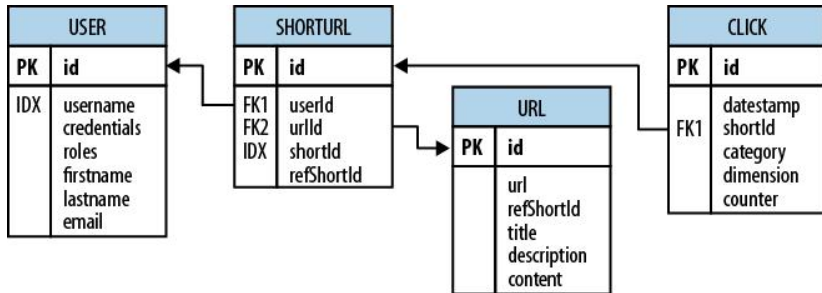
## But

Service permettant d'assigner une clé unique de quelques caractères à une URL d'une page web spécifique

## Spécifications

- Une clé est renseignée par un utilisateur du service
- Une clé est associée à une URL
- Chaque click sur une URL réduite redirige automatiquement vers l'URL réelle
- Un utilisateur doit pouvoir se loguer pour gérer ses URLS réduites et connaître en temps réel l'usage (quotidien, mensuel, ...) qui en est fait

# Exemple illustratif : Modélisation relationnelle



# Exemple illustratif : Version NoSQL

Table: shorturl		
Row Key:	shortId	
Family:	data:	Columns: url, refShortId, userId, clicks
	stats-daily: [ttl: 7days]	Columns: YYYYMMDD, YYYYMMDD\<country-code>
	stats-weekly: [ttl: 4weeks]	Columns: YYYYWW, YYYYWW\<country-code>
	stats-monthly: [ttl: 12months]	Columns: YYYYMM, YYYYMM\<country-code>

Table: url		
Row Key:	MD5(url)	
Family:	data: [compressed]	Columns: refShortId, title, description
	content: [compressed]	Columns: raw

Table: user-shorturl		
Row Key:	username\<shortId	
Family:	data:	Columns: timestamp

Table: user		
Row Key:	username	
Family:	data:	Columns: credentials, roles, firstname, lastname, email

- La table d'URL courtes :
  - stocke directement les compteurs par période
  - Chaque période est une famille de colonne
  - Chaque nom de colonne dans une période est une date
- La table d'URL
  - sépare en deux familles (comprimé), les méta-données de la page et son contenu
- La table user-shortid :
  - lie le nom d'un utilisateur avec une URL-courte
  - permet une recherche plus rapide des relations User-ShortURL



## 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

# 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)

- Table de hachage distribuée :

- 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 x 2 = stockage et puissance de calcul x2

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		
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 caractère
- 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
<b>Rowkey1</b>	vers2	<b>val1</b>	vers1	<b>val6</b>	vers1	<b>val7</b>	vers3	<b>val10</b>	
							vers2	<b>val11</b>	
	vers1	<b>val2</b>					vers1	<b>val12</b>	
Rowkey2	vers1	<b>val3</b>					vers1	<b>val13</b>	vers1 <b>val16</b>
Rowkey3			vers1	<b>val5</b>	vers1	<b>val9</b>	vers1	<b>val14</b>	vers1 <b>val15</b>

## Définition

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

# ColumnFamily

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 différente version de la cellule (ou timestamp)

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

## 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 version que l'on peut stocker 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

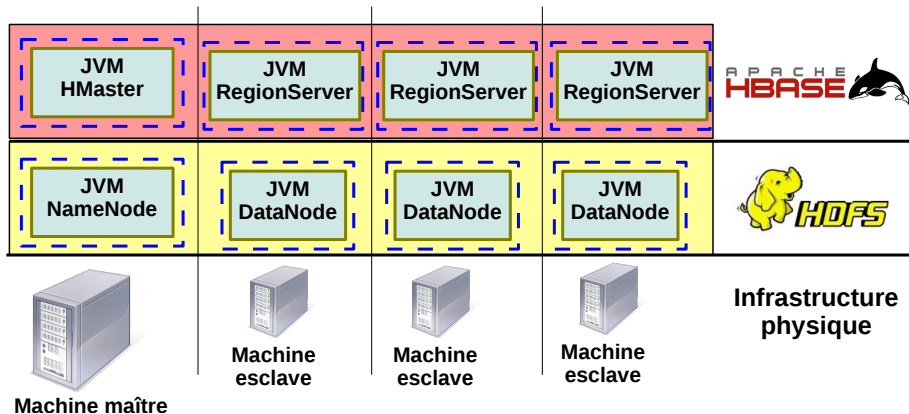
## Définition

- 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

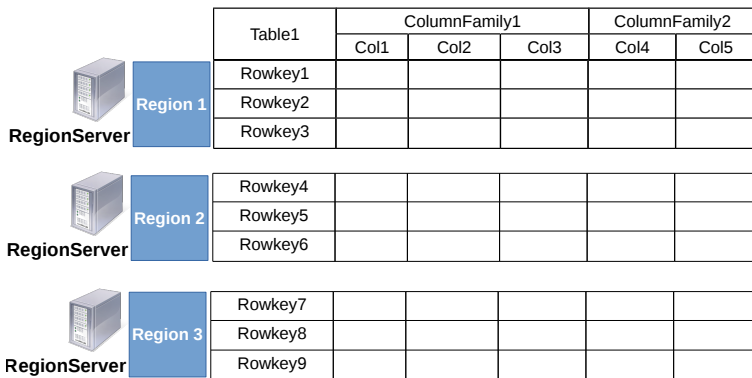


## Une architecture maître esclave

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

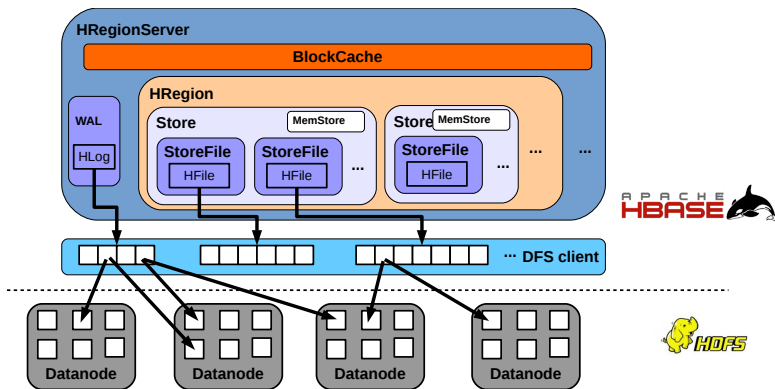


# Découpage des données sur le cluster



## 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

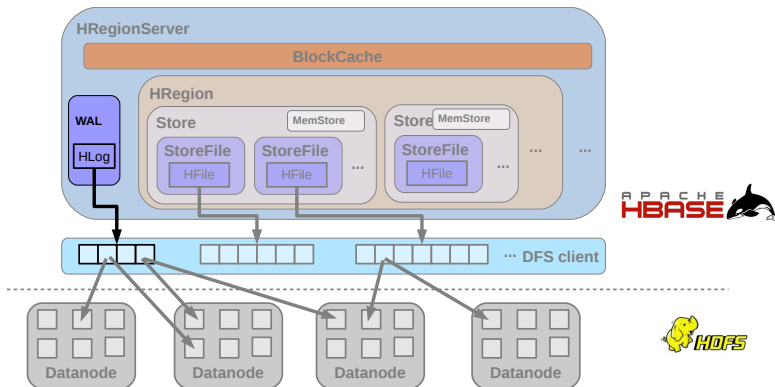


## Caractéristiques principales

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



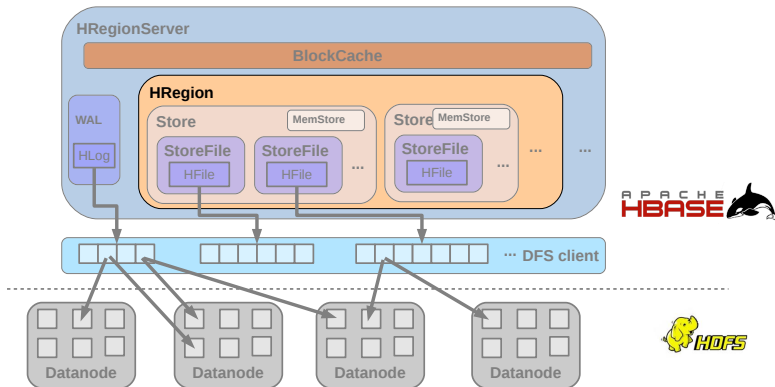
# RegionServer : Write Ahead Log (WAL)



## 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

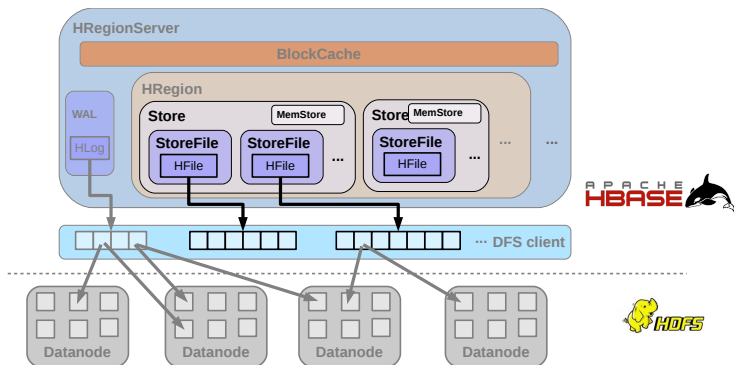
# RegionServer : HRegion



## Caractéristiques des HRegion

- Représente une région
- Gère un sous ensemble 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 du découpage des tables

- Une table est segmentée en plusieurs partitions
- Une partition d'une table est gérée par une Region
- Un RegionServer gère plusieurs Region
- Une Region contient plusieurs Store
- 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

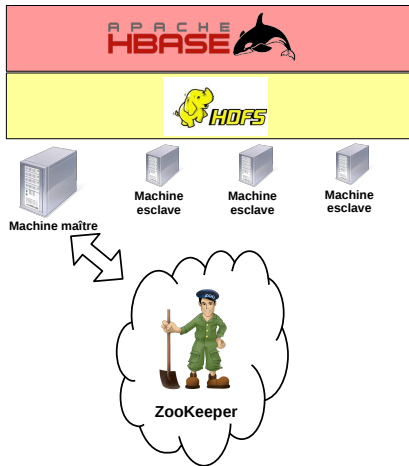


## Caractéristiques

- Logiciel Apache de gestion de configuration pour systèmes distribués
- Les données sont stockées en espace de noms hiérarchique

## Rôle pour HBase

- Surveille l'état du cluster et en informe régulièrement le HMaster
- Stocke la localisation de la table hbase :meta dans le cluster



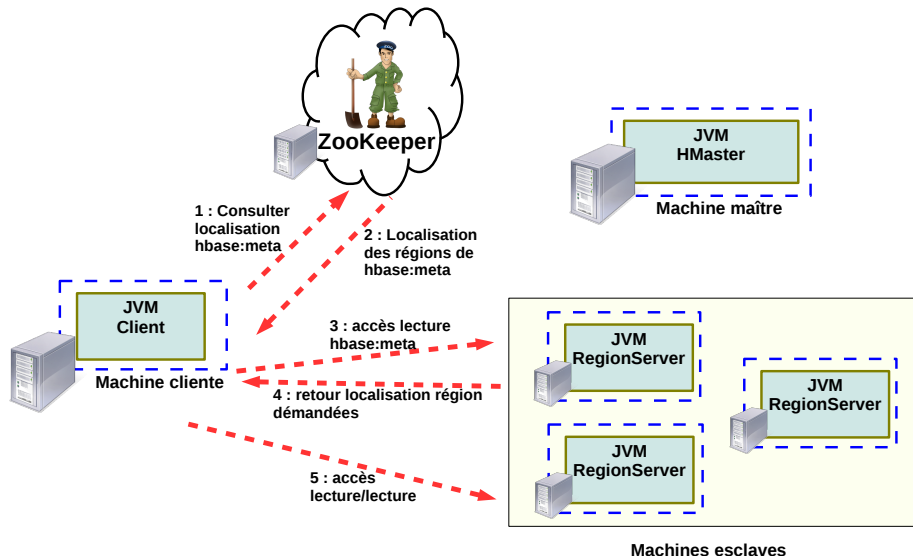
## Caractéristiques

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

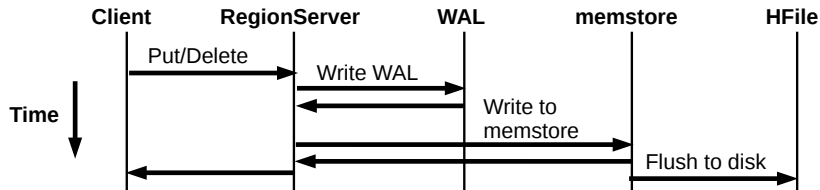
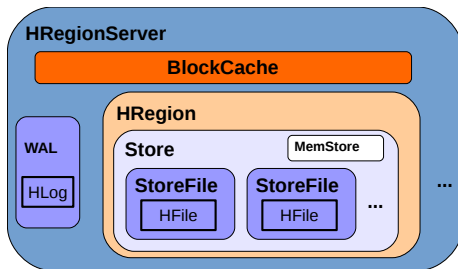
hbase

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
```

## Création/fermeture d'un descripteur de table

```
Connection c = ..;
TableName tableName = TableName.valueOf("ma_table");
Table table = connection.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

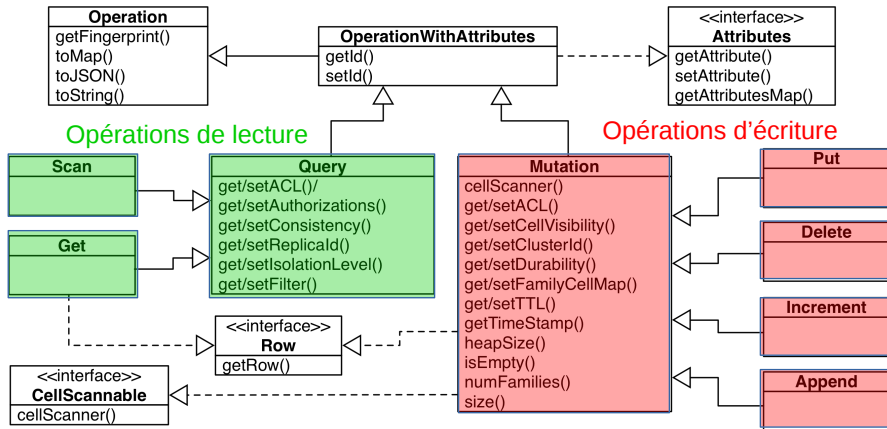
### Vers le binaire

```
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);
...
```

# Opération sur les tables



# Mutation : les Put

## 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

# Query : les Get

## 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 toute 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));
```



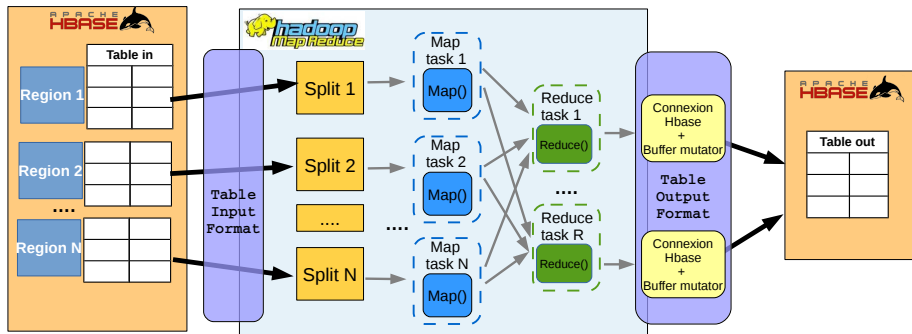
## Caractéristiques

- 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 avec Hadoop MapReduce



## Interfaçage Hadoop-Hbase

- Une région = Un split
- Un reduce = Un client Hbase

## InputFormat spécifiques à Hbase

- `TableInputFormat` :
  - type clé : `ImmutableBytesWritable`, représente le rowkey en binaire
  - type valeur : `Result`, résultat issu d'un Scan
  - deux paramètres nécessaires : le nom d'une table et un scan
- `MultiTableInputFormat` : similaire mais pour plusieurs tables/scans
- `WALInputFormat` : Accéder au WAL de Hbase.

## Classe utilitaire pour configurer un `TableInputFormat`

```
class TableMapReduceUtil{
...
static void initTableMapperJob(String table, // nom table
    Scan scan, // scan de la table
    Class<? extends TableMapper> mapper, // type mapper
    Class<?> outputKeyClass, // type clé sortie map
    Class<?> outputValueClass, //type valeur sortie map
    Job job)//job map reduce
...
}
```

## Mapper pour table Hbase

- Héritage de la classe `TableMapper<KEYOUTMAP, VALOUTMAP>`
- Redéfinition de la méthode

```
public void map(ImmutableBytesWritable key, Result value  
                , Context context );
```

## Reducer pour écrire dans une table Hbase

- Héritage de la classe `TableReducer<KEYIN, VALUEIN, KEYOUT>`
- La valeur de sortie est une sous classe de `Mutation` :

Uniquement `Put` ou `Delete`

## OutputFormat spécifiques à Hbase

- `TableOutputFormat` :
  - type clé : `ImmutableBytesWritable`, représente le rowkey en binaire
  - type valeur : `Mutation`
  - un paramètre nécessaire : le nom d'une table
- `MultiTableOutputFormat` : similaire mais pour plusieurs tables

## Configurer un `TableOutputFormat`

```
job.getConfiguration().set(TableOutputFormat.OUTPUT_TABLE, "tab_out")
job.setOutputFormatClass(TableOutputFormat.class);
job.setOutputKeyClass(ImmutableBytesWritable.class);
job.setOutputValueClass(<sous classe de Mutation>);
```

ou bien les méthodes statiques `initTableReducerJob` de la classe

`TableMapReduceUtil`

# Codage Map-Reduce sur Hbase : squelette

```
public class HbaseMapper extends TableMapper<KOUTM, VOUTM>{  
    void map(ImmutableBytesWritable key, Result value, Context c){...}  
}
```

```
public class HbaseReducer extends TableReducer<KINR, VINR, KOUTR>{  
    void reduce(KINR key, Iterable<VINR>, Context c){...}  
}
```

```
Configuration conf = HBaseConfiguration.create();  
//DESACTIVER LA SPECULATION  
Job job = Job.getInstance(conf, "job");  
//config table entrée  
Scan scan = ....;  
TableMapReduceUtil.initTableMapperJob("table_in", scan,  
    HbaseMapper.class, KOUTM.class, VOUTM.class, Job job);  
//config table sortie  
job.getConfiguration().set(TableOutputFormat.OUTPUT_TABLE, "t_out");  
job.setOutputFormatClass(TableOutputFormat.class);  
job.setOutputKeyClass(ImmutableBytesWritable.class);  
job.setOutputValueClass(<sous classe de Mutation>);
```

## 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 stockée sur disque
- 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