2018/2019

FOCUS SUR HBASE

RAPPELS

- La base de données Hadoop
- Développée à partir du papier BigTable de Google, en 2006
- No SQL
- HBase est une Base de données orientées colonnes **distribuée** qui utilise HDFS pour stocker ses données.
- Gains en lectures et écritures => Real-Time Query Data
- Architecture Fault Tolerant, distribuée, hautement scalable

HBase Table

- RowKey
- Column Family => Tous les membres ont le même préfixe.
- Colonne
- TimeStamp, les cellules sont versionnées par un timestamp
  => Permet l'update sur HDFS: on append une valeur ayant la même rowkey, avec une version plus récente (timestamp)

La valeur d'une cellule est un tableau de bytes non interprété.

Atomicité de l'insertion/Update de Rows.

HBase Table

KeyValue = coordonées

(Row + Family + Qualifier + Timestamp + Value)

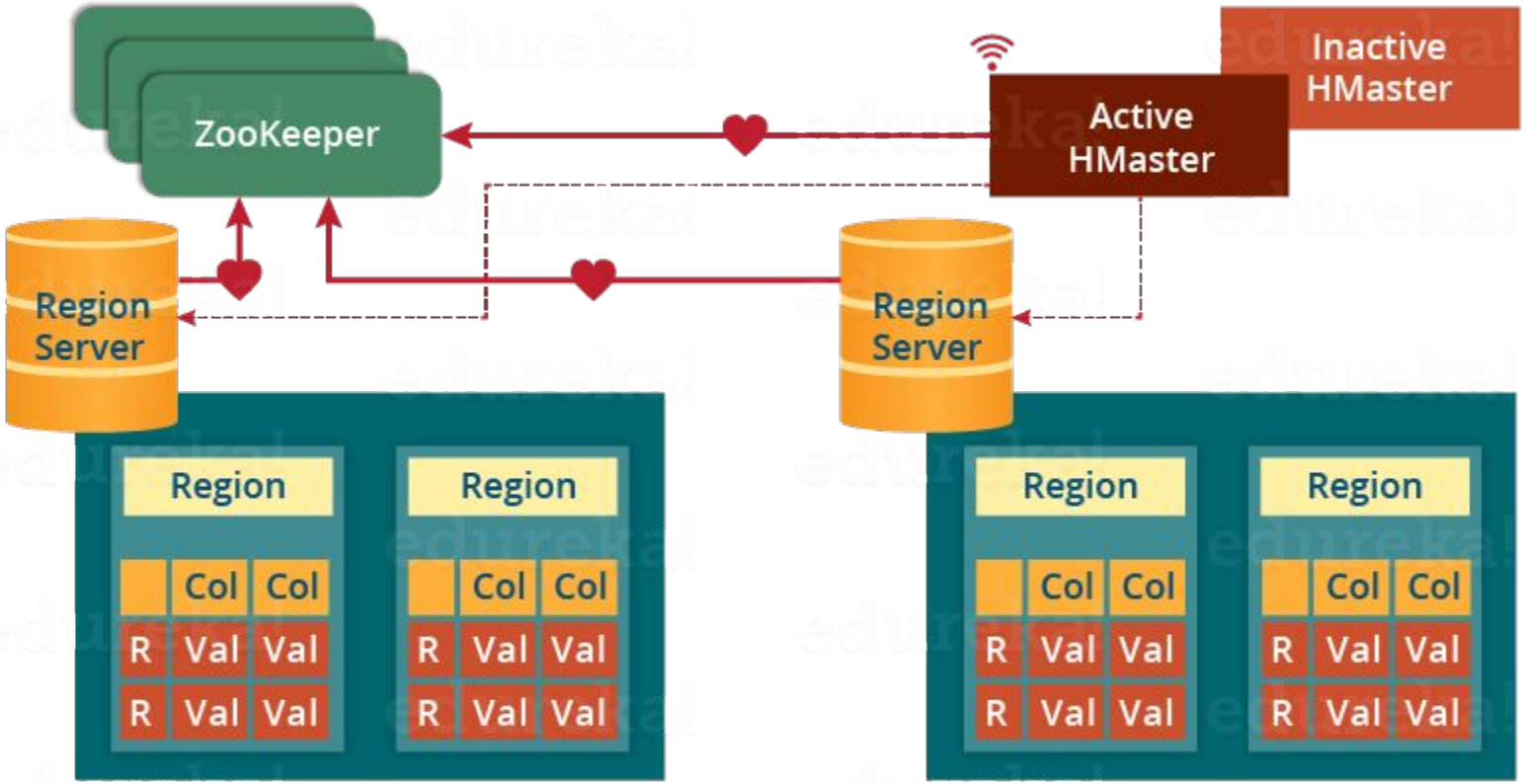
Timestamp

= version

Composants de HBase

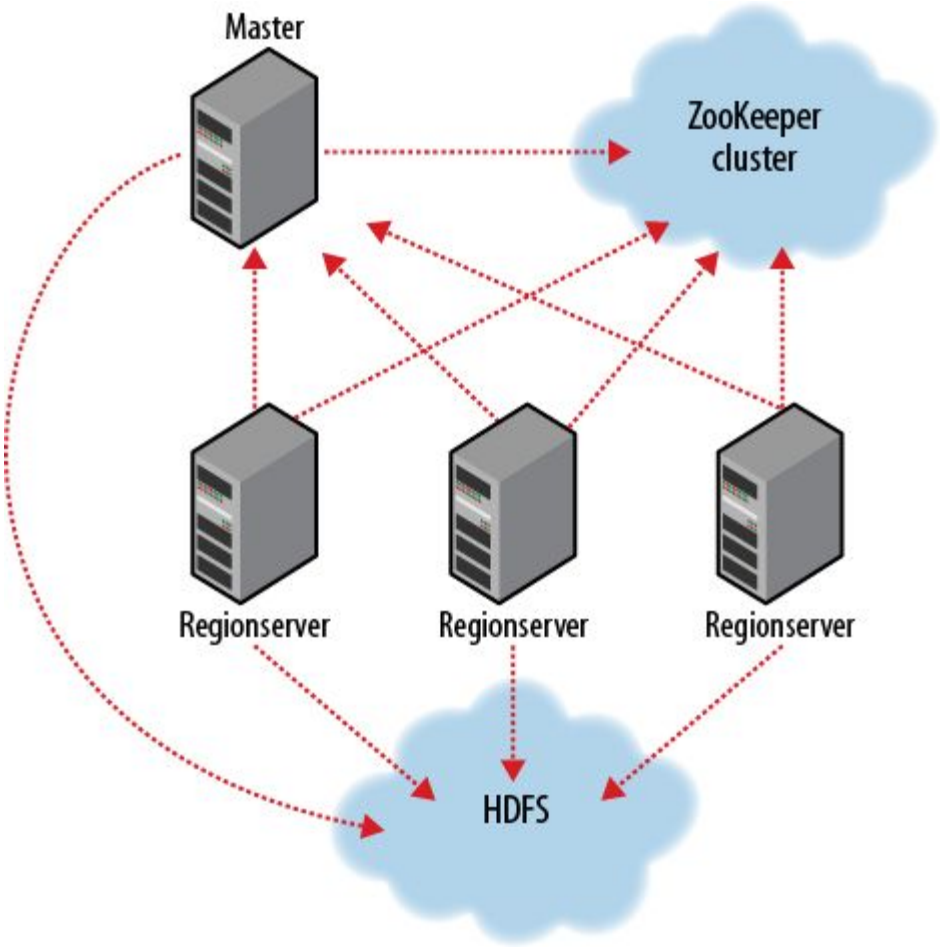
- HMaster
  - Contient les Metadatas
  - Gère les regionServers
  - Load Balancing de régions pour répartir la charge
- RegionsServers
  - Contiennent les données des tables, réparties en partitions: les régions
  - Les Regions sont réparties sur plusieurs server: les RegionServer
  - Responsables des lectures / écritures
- Regions: Partition d'une table délimités par des rowkeys => contient les données de la première RowKey A (inclusive) à la dernière RowKey F (exclusive).
- L'assignement des Regions est réalisé via Zookeeper.
- Une table n'a initialement qu'une seule région. A partir d'une taille limite, grâce à l'auto-splitting, la table est ensuite divisée en 2 régions de taille à peu près égales.
-

HBase Architecture



Zookeeper au sein de HBase

- Adresse du Master et des backup Masters
- Liste des régions non assignées
- Adresse du serveur qui stocke les métadonnées des régions
- L'état des tables lors de l'assignement des régions
- Informations lors du log-splitting



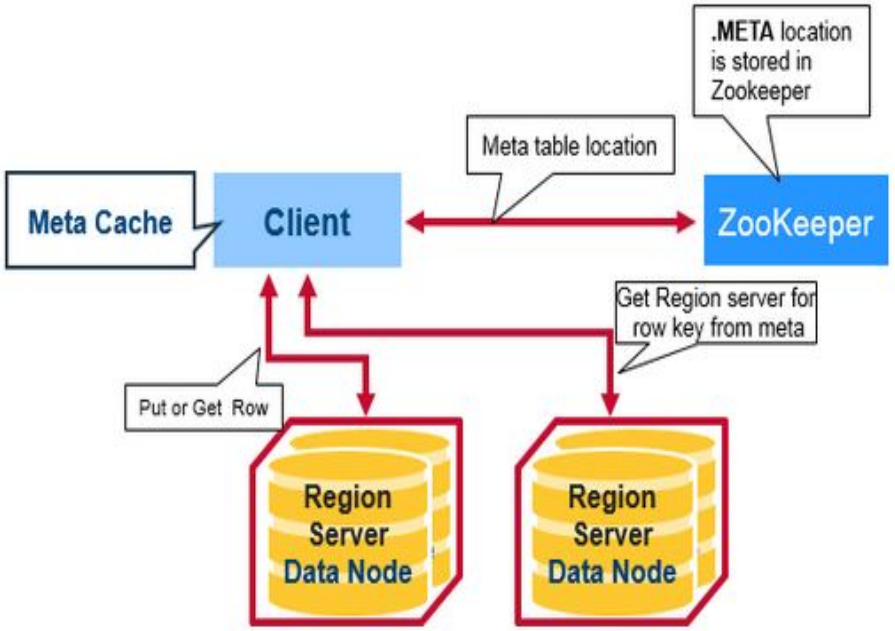
Read Path

1. Demande à Zookeeper l'adresse du serveur qui contient les métadonnées des régions

2. Demande au serveur la région et son emplacement pour une rowkey ou d'un range de rowkey définis

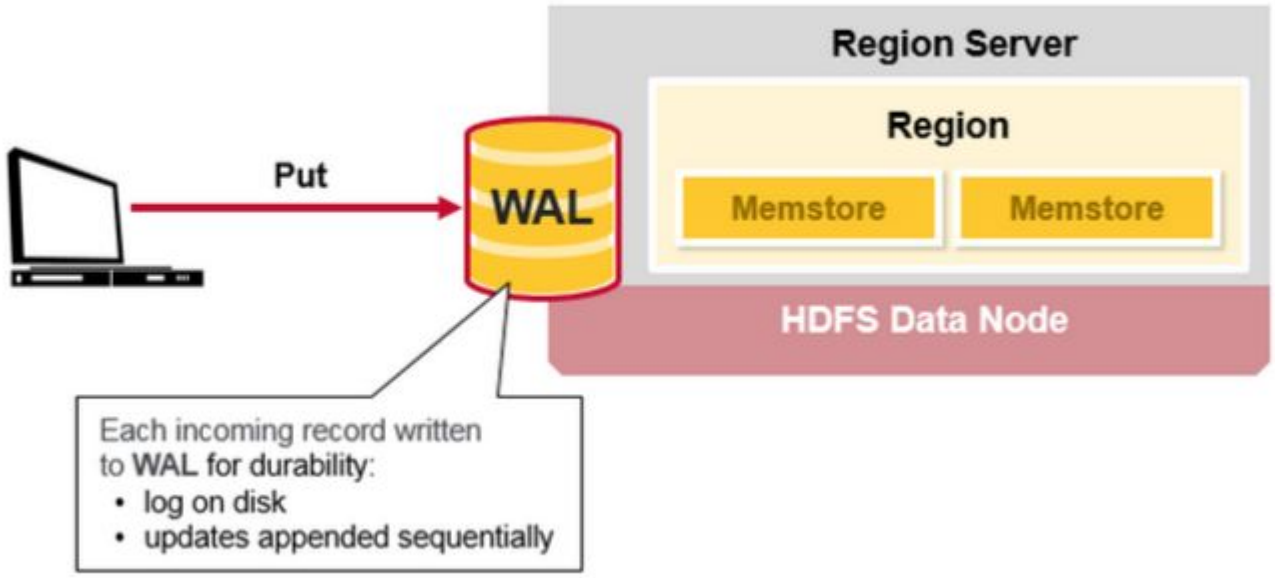
3. Une fois l'adresse du région serveur obtenu, requête directement celui-ci

4. Conserve en cache (côté client) l'adresse du meta-region-server



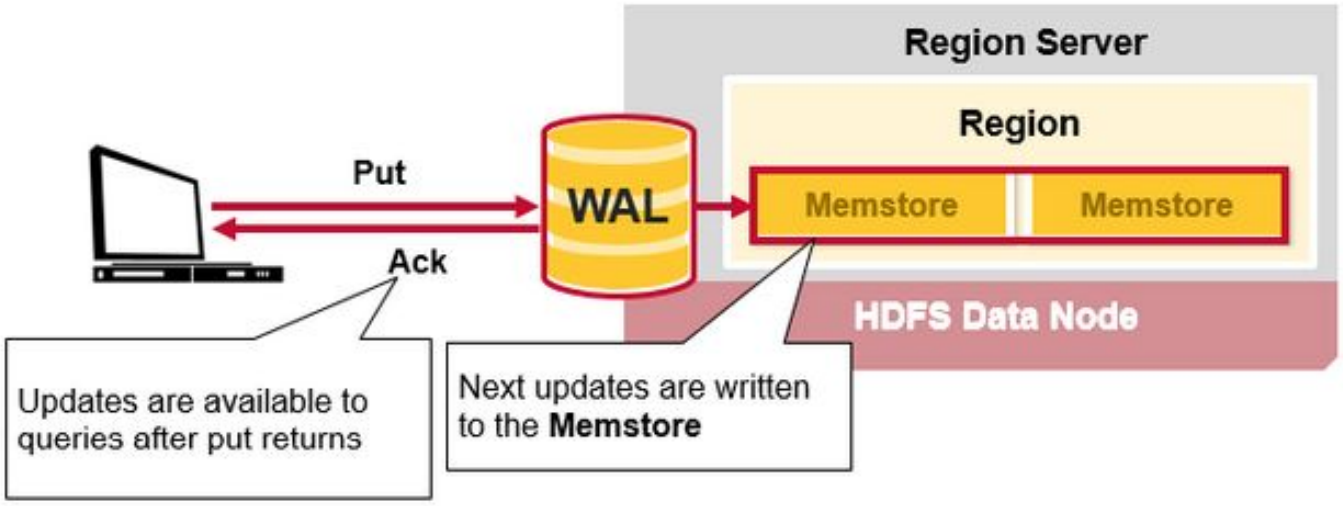
Write Path Put/Delete

1. Etapes précédentes (1-2) pour récupérer le bon région serveur, et envoyer la requête
2. Ecriture dans le WAL (Write Ahead Log) de façon séquentielle
   => Permet de stocker les données non persistées en base sur HDFS, afin de les rejouer en cas de failure.



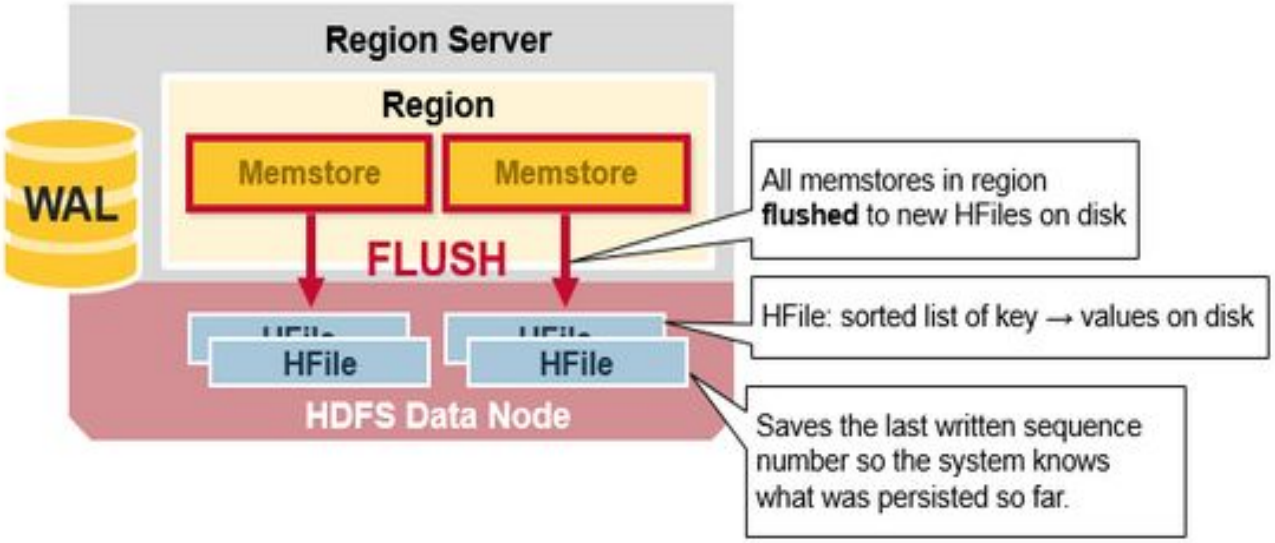
Write Path Put/Delete

3. Ecriture dans le Memstore: les données sont triées en fonction de leur rowkey et stockées en mémoire.
4. Retour OK au client: la donnée n'est pas encore stockée définitivement sur HBase mais elle est déjà persistée sur les WALs.



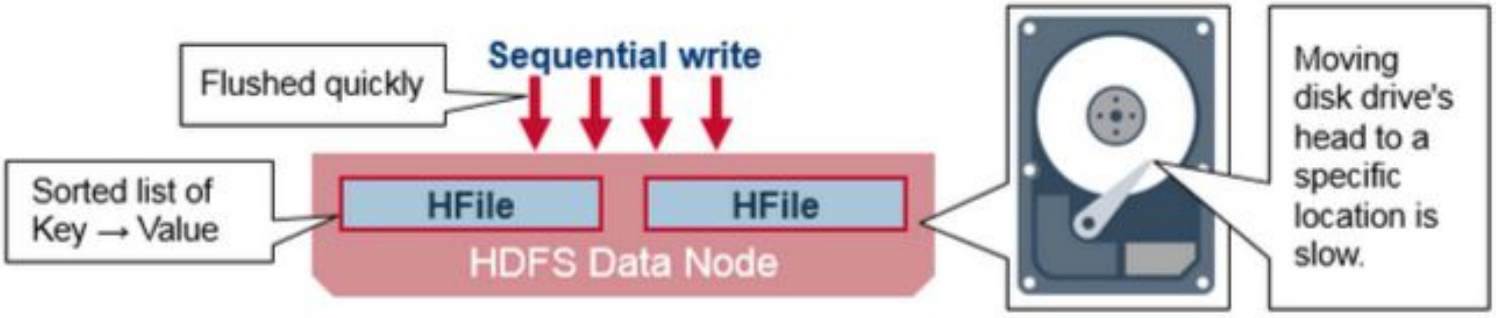
HBase Flush

Ecriture des données sur HDFS via des HFiles lorsque le memstore est full
1 memstore et 1 HFile par Column Family
La taille des HFiles est configurable, et représente un pourcentage de la taille d'un bloc HDFS. Une taille minimum est configurée par défaut afin d'éviter trop de petits fichiers



HFile

Ecriture séquentielle afin d'optimiser les performances: le tri se fait en mémoire, et tout le fichier est écrit une fois



Key | Value

Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key | Value

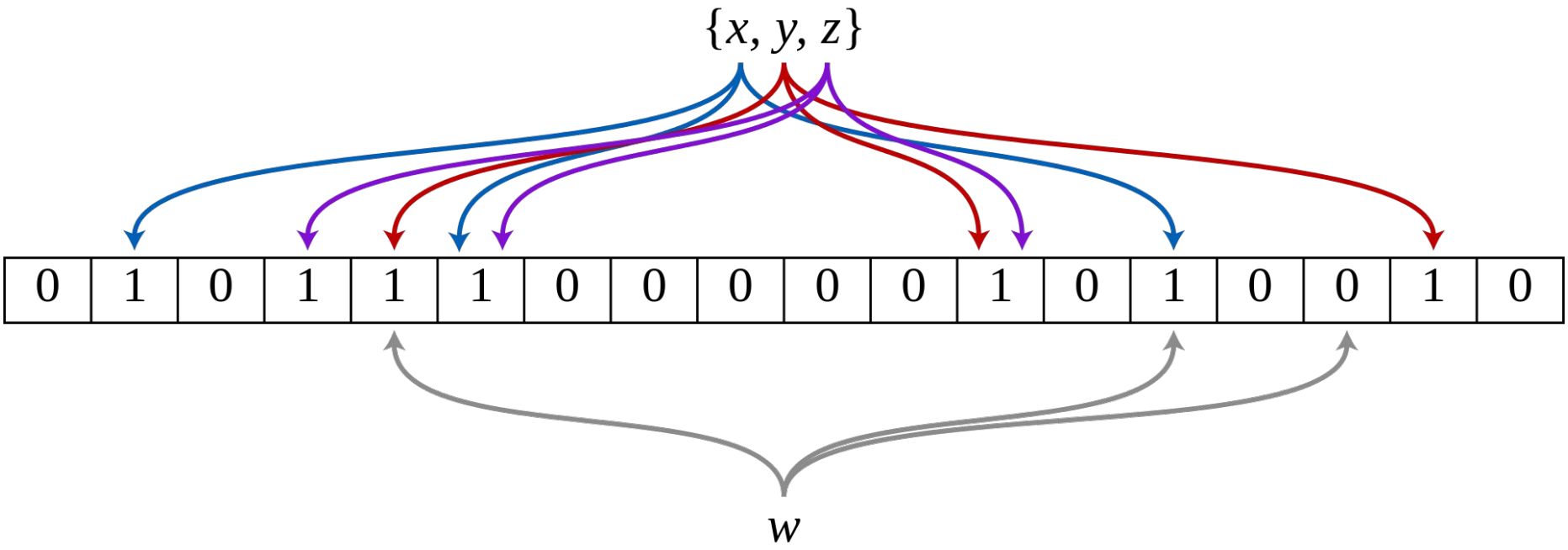
Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

Bloom Filters:

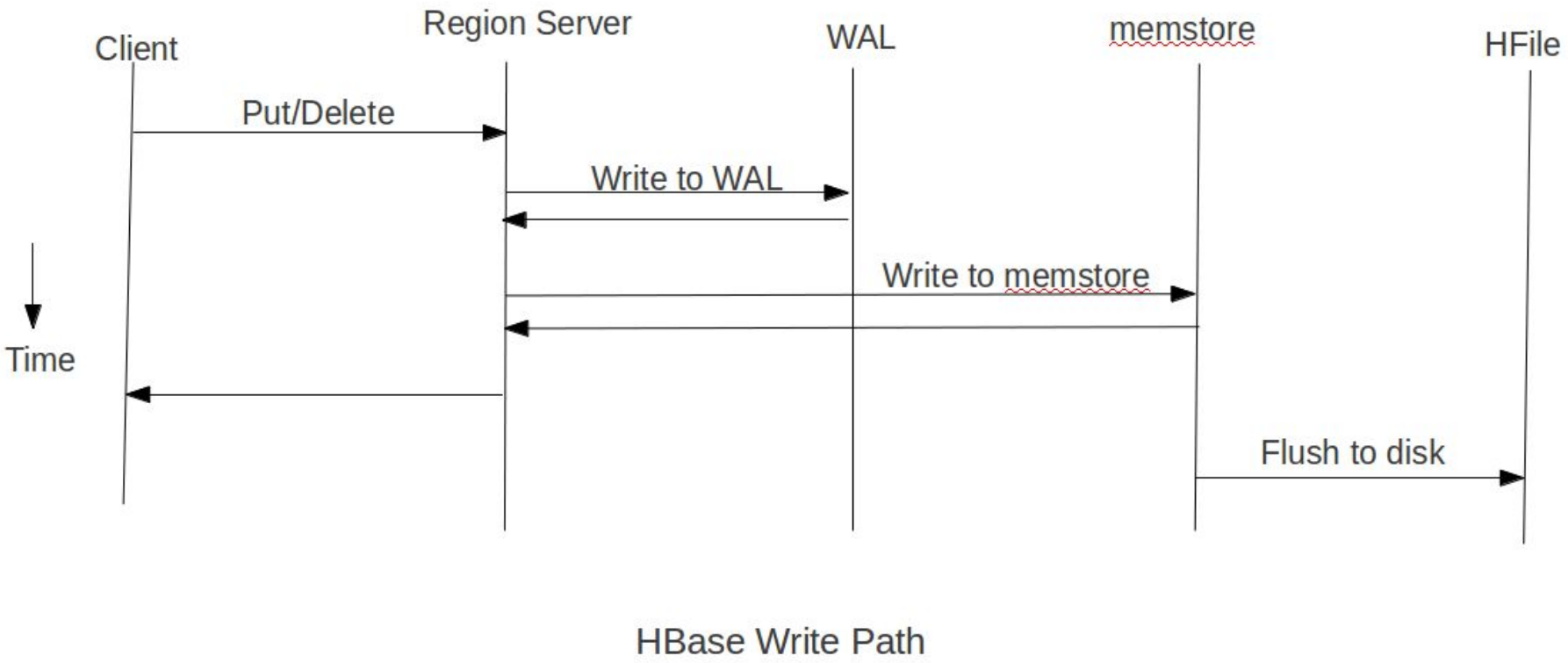
Structure de données qui permet de savoir:

- avec certitude que l'élément est absent de l'ensemble (il ne peut pas y avoir de faux négatif)
- avec une certaine probabilité que l'élément peut être présent dans l'ensemble (il peut y avoir des faux positifs)

Très utile pour éviter de lire des blocs de données => ORC ou Parquet



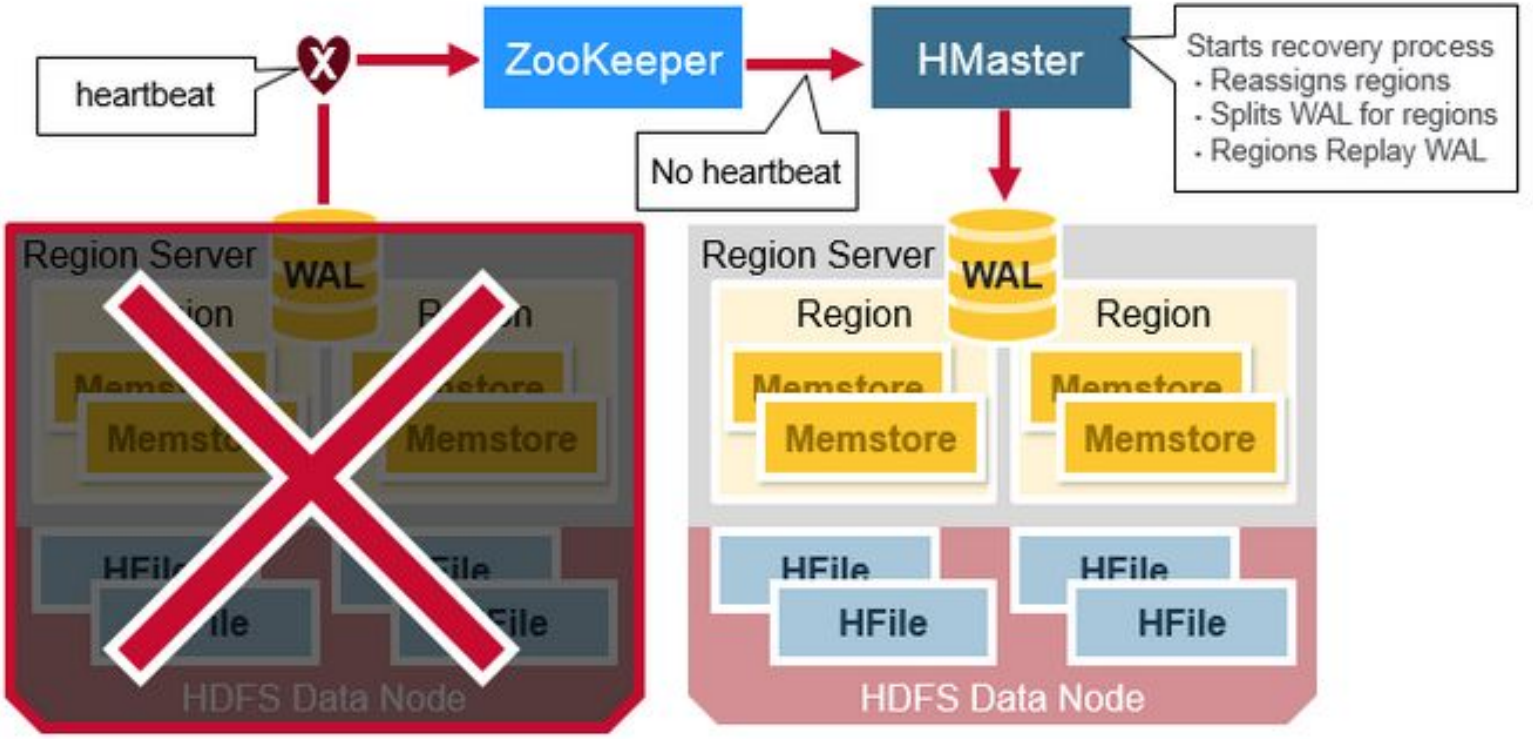
HBase Write Global Overview



HBase Write Path

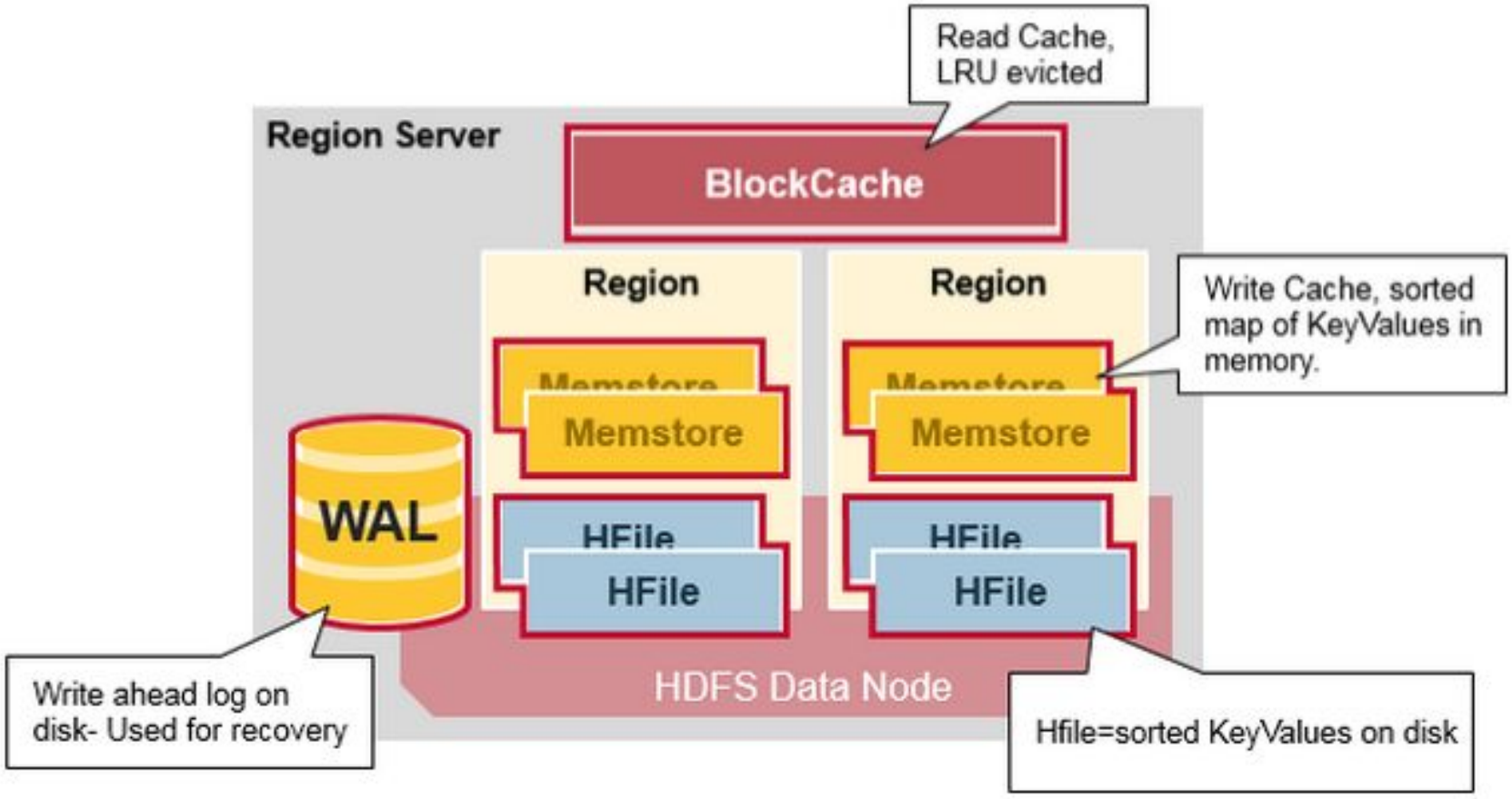
HBase RegionServer Crash Recovery

- HMaster est notifié par Zookeeper car il n'y a plus de HeartBeat
- Découpe du WAL et Réassignement des régions sur les régions serveurs disponibles
- Reprocess du WAL: Lecture, tri en mémoire (Memstore) et Flush



HBase RegionServer

Les données sont réparties par range de rowkeys sur les régions serveurs
Les Memstores et le BlockCache en mémoire
Les HFiles et les WALs sont stockés sur HDFS



HBase RowKey Design

- Les rowkeys sont stockées de façon lexicographique: les rowkeys qui se ressemblent sont stockées ensemble, dans la même région.

- Distribution uniforme (type Hash) est fortement conseillée pour une répartition optimum.

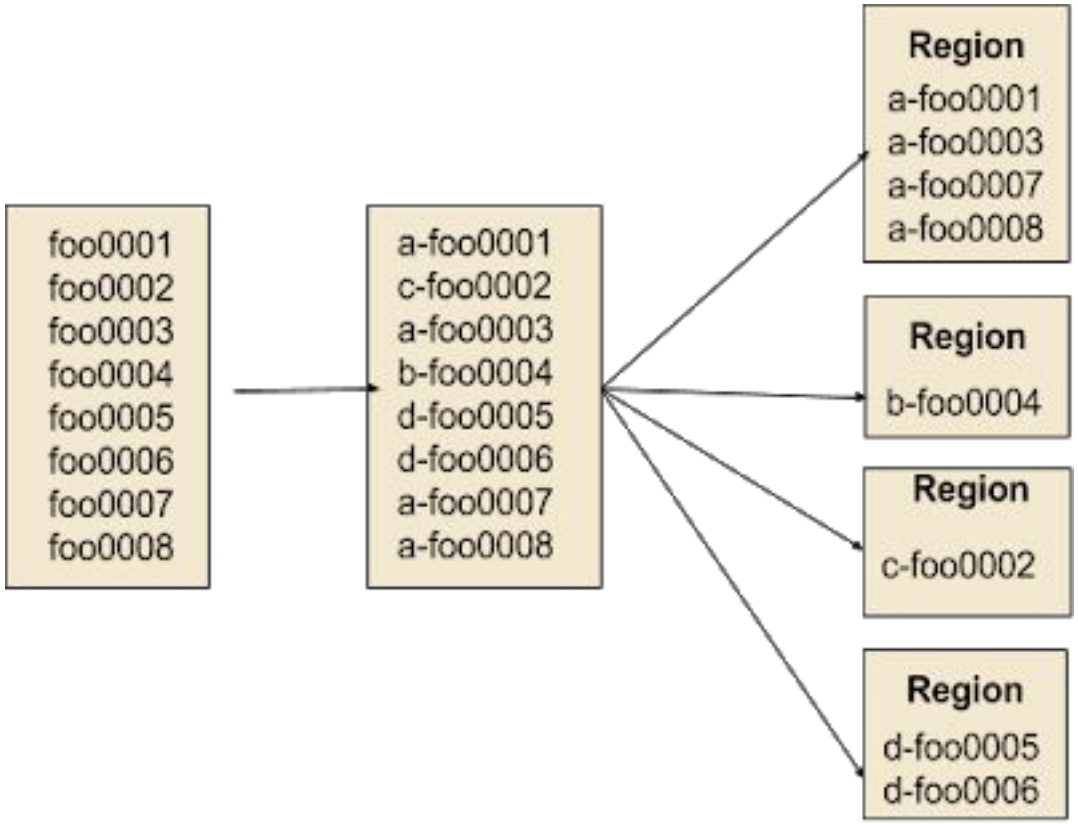
- Eviter le **HotSpotting:** Toutes les clés et toutes les requêtes sont orientées vers le même région serveur.

- En cas de **HotSpotting**, le traffic est trop important pour le serveur ciblé et il en résulte de mauvaises performances, voire des indisponibilités de régions serveurs.

- Le Design des rowkeys est très important afin de garantir une stabilité du cluster, de l'application client, et des performances optimums.

HBase RowKey Design

- Hash de la clé: Distribution uniforme
- **Salting Préfix:** on par exemple créer autant de préfixes que de régions, et les assigner de façon random
- Reverse TimeStamp



Hadoop Overview