BDD Cassandra

Introduction :

- Base de Données pour la Big Data
- Fait pour les infrastructures avec des millions de requêtes ou une infrastructure cloud
- Utilisation : Github, Instagram, Netflix, Twitter
- Basé sur le NoSQL, orienté colonnes
- Equivalents : HBase, Apache Accumulo
- Développé à l'origine par des ingénieurs de chez Facebook en 2007
- Devenu Open Source appartenant à la fondation Apache en 2009
- Reprend les concepts de 2 BDD existantes
    - BigTable créé par Google pour son modèle de données orienté colonne et son mécanisme de persistance sur disque
    - Dynamo créé par Amazon pour son architecture distribuée sans noeud maître
- *Objectifs :*
    - Gérer une grande quantité de données
    - résister aux pannes
    - être performante
- *Avantages*
    - Très rapide pour manipuler un volume important de données
    - schémas flexibles grâce à sa représentation en colonnes
    - Evolution facile dans un environnement distribué
    - Intègre des mécanismes de réplication de données
    - Accès disque par des accès séquentiels et non aléatoire pour éviter les latences

Fonctionnement :

- Cassandra fonctionne avec des Clusters
- Clusters = regroupement de noeuds (ou serveurs physiques) qui communiquent entre eux (protocole peer-to-peer)
    - Tous les noeuds sont égaux = pas de noeud maître ou un processus centralisant leur gestion

Théorème CAP :

- paramètres sur lesquels on peut jouer pour configurer une base de données distribuée
    - Cohérence (C)
    - Disponibilité (A)
    - Tolérance aux pannes (P)
- Le théorème postule sur le fait que l'on ne peut choisir que 2 paramètres sur 3
    - CA (non résistante aux pannes)
    - CP (non disponible à 100%)
    - AP (non cohérente à 100%)

Architecture :

- Organisée en 3 couches
    - API (reçoit les requêtes des clients au format Thrift)
    - Dynamo (distribution des données entre les différents Noeuds)
    - Base de données (persistance des données sur le disque)

Modèle des données :

- Dans une base relationnelle
    - Composée de colonnes (plus petite unité de donnée)
        - nom : 64ko Max
        - Valeur : 2Go Max
- Dans Cassandra
    - Composée de lignes (row) (nom + colonnes (nom + valeur)
        - nom clé : 64ko Max
        - nb max de colonnes : 2 milliards
    - Le nom de la colonne est une valeur
    - les colonnes sont triées par leur nom
    - Possibilité de faire un lien entre deux entrées
    - Valueless column : possibilité que le nom de la colonne se suffise à lui seul. Cela permet d'optimiser les performances
    - deux usages pour les rows
        - Wide Row : contient des milliers, millions de colonnes (timeline twitter)
        - Skinny Row : en contient moins

Modèle Physique de données :

- *Keyspaces et Tables*
- Dans un cluster Cassandra on trouve :
    - Des Tables
    - Des Keyspaces (ressemble à une base)
- Dans un modèle multi-tenant, on sépare les données de chaque partie dans des keyspaces
- Dans chaque Keyspace on trouve des tables
- Les tables sont pareil qu'en SQL (Lignes et colonnes)
- **Cluster** *1←→ n* **Keyspace** *1← → n* **Table**

- *Partitions, Colonnes et Cellules*
- Dans une table les données sont stockées sous forme de lignes et de colonnes.
- Les données sont disposées sur des lignes (partitions)
- Dans une partition, on trouve une série de colonnes = clé/valeur
- Une partition peut contenir 1 à N colonnes (n = 2 milliards)
- **<Clé de partition, <Clé de colonne,cellule>>**

- *Distribution des données*
- Dans un cluster, les partitions dans une tables sont réparties entre plusieurs noeuds
- Une partition se trouve toujours entièrement sur un seul noeud.
- Choix du type de partitionnement se fait au niveau du keyspace

    - BytesOrderedPartitioner
    - RandomPartitioner (ou Murmur3Partitioner pour Cassandra V2)
- 2 types de répartition possible :
    - De manière ordonnée, chaque noeud prend en charge une plage de clé de partition triée par ordre croissant
        - Converse l'ordre, mais noeuds non équilibrés
        - **Exemple :**
            - Clé de partition = nom de famille
            - Noms entre A et E dans le noeud 1
            - Noms entre F et J dans le noeud 2
            - ….
    - De manière aléatoire, chaque noeud prend en charge une plage de la clé de partition distribuée uniformément
        - Conseillé pour mieux répartir les charges

- *Tokens*
- Moyen simple de répartir les données sur le cluster.
- Permet de distinguer les partitions
- On répartit uniformément un intervalle (plage) à chaque noeud.

- *Le Noeud Coordinateur*
- Dans cassandra pas de notion maître/esclave
- Garantie l'absence de point unique de défaillance
- Le client fournit toujours une partition.
- Si le hash de la partition ne correspond pas à sa plage, il va rediriger la requête vers un autre noeud, responsable de la donnée
- Permet la redirection de la requête
- Tous les noeuds peuvent jouer le rôle de coordinateur

Structure d'un Table :

- Les tables sont fortement typées
    - Clé de partition (#partition)
    - Clé de colonne (#col)
    - Cellule
- Exemple de création d'une table avec l'API Thrift
    - create column family user
      with key_validation_class = LongType
      and comparator = UTF8Type
      and default_validation_class = UTF8Type
- Exemple de requête avec l'API Thrift

        - get(#partition,#col)
        - Avec getSlice il est possible de récupérer une tranche de colonnes pour une partition donnée
            - getSlice(#partition, #col_start, #col_end, reverse, limit)
        - Avec multiGetSlice, il est possible d'exécuter une requête sur une liste de partitions. Permet une simplicité côté client car une seule requête. Par contre le temps de réponse du temps du temps de réponse de la plus longue requête.
            - multiGetSlice(list #partition,#col_start,#col_end,reverse,limit)
        - Insérer une cellule dans une partition
            - insert(#partition,#col,cellulle)
        - Supprimer la colonne identifiée
            - remove(#partition,#col)
        - Il est important de définir la limite d'une requête afin de ne pas surcharger
- Résumé des méthodes API Thrift
    - Accès direct à une cellule (get, insert, remove)
    - Accès à une plage de colonnes (getSlice)
    - Accès à une plage de colonnes pour un ensemble de partitions (multiGetSlice)

Ecriture des données :

- Les données sont écrites dans un premier temps dans un journal de commit (pour s'assurer de la durabilité), puis dans une structure en mémoire appelée la Memtable. L'enregistrement est effectué quand ces deux étapes ont réussit
- Tunable Consistency : permet au client de s'assurer que la donnée est écrite et répliquée (même système pour la lecture)
- Le client indique le Consistency Level désiré :
    - One : je veux être certain qu'au moins une replica a été persistée durablement
    - Quorum : je veux être certain que la majorité des replicas ont été persitées durablement
    - All : Je veux être certain que toutes les replicas ont été persistées durablement.
- Lorsqu'un noeud est indisponible le noeud coordinateur écrit la donnée sur un autre noeud , et va rejouer le write sur le noeud qui était indisponible quand il revient dans le ring
- 3 colonnes pour l'écriture
    - nom
    - valeur
    - timestamp (le plus récent fait foi)

Performances en écriture :

- En écriture, Cassandra est extrêmement rapide par :
    - Écrit dans le fichier Commit Log
    - Écrit en mémoire
    - Pas d'update

Performance en lecture :

- En lecture Cassandra est quasiment aussi performant
    - Contact en priorité le noeud le plus proche de la donnée
    - Index pour row Key et col.name + Cache
    - Bloom Filter : structure utilisée pour vérifier si la clé d'une ligne existe lors d'une requête

Snitch :

- Une IP → Un endroit
- Utile pour répartir les réplicas entre les racks, entre les DataCenters
- Le "Dynamic snitch" prend aussi en compte le temps de réponse des noeuds

Gossip :

- Le fait de savoir qu'un autre noeud revient dans le ring
- Chaque seconde, chaque noeud contacte entre 1 et 3 noeuds (peer-to-peer)
- Périodiquement, les noeuds vont échanger des informations sur leur état mais aussi sur ce qu'ils savent des autres noeuds.
- Toutes les secondes chaque noeud va échanger des messages avec 3 noeuds maximum. De plus, une version est associée à ces messages pour permettre d'écraser les informations les plus anciennes
- En cas d'indisponibilité d'un noeud, les autres vont stocker temporairement les données jusqu'à ce qu'il soit de nouveau disponible.

Répartition des données :

- Réplication : processus permettant de stocker des copies des données sur de multiples noeuds afin de permettre leur fiabilité et la tolérance à la panne.

CQL 3

- CQL = Cassandra Query Language
- Pour la manipulation des données cela fonctionne comme en SQL
    - Créer une Table :
        - CREATE TABLE user(
            user_id bigint,
            firstname text,
            lastname text,
            age int,
            PRIMARY KEY(user_id));
    - Insérer une entrée
        - INSERT INTO user(user_id, firstname, lastname, age) VALUES(10, 'Jean', 'MARTIN', 33);
    - Mettre à jour une entrée
        - UPDATE user SET age = 33 WHERE user_id = 10;
    - Supprimer une entrée
        - DELETE FROM user WHERE user_id=10;
- Par contre il n'y a pas de JOINTURES (ce n'est pas une base relationnelle).
    - Il faut faire un travail en amont sur la modélisation des tables
- Pas de GROUP BY
    - Il faudra gérer les agrégations côté client

- Clause WHERE Limitée
    - Possibilité d'égalité, inégalité
    - Impossible d'utiliser le *to_char*, *to_date* ou *like*
- Clause ORDER BY Limitée
    - Possibilité d'utiliser un order by mais seulement sous certaines conditions
- Pas de CONTRAINTES
    - Il n'existe pas de primary key
- Dualité de INSERT/UPDATE
    - Techniquement, dans Cassandra, le INSERT et UPDATE sont traduit par UPSERT. La différence entre les 2 opérations est purement cosmétique.

https://www.infoq.com/fr/articles/modele-stockage-physique-cassandra
http://www-igm.univ-mlv.fr/~dr/XPOSE2010/Cassandra/cassandra.html
http://blog.wmaker.net/Cassandra-la-technologie-NoSQL-inventee-par-Facebook_a1190.html
http://www.journaldunet.com/developpeur/outils/comparatif-des-bases-nosql/cassandra.shtml
http://soat.developpez.com/articles/cassandra/#