Starter-Kit Cassandra

Lexique :

Cluster : regroupement de noeuds (ou serveurs physiques) qui communiquent entre eux (protocole peer-to-peer)
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 = Cassandra Query Language.

Introduction :

Aujourd'hui, pour réaliser un site internet dynamique, un application, une base de donnée est nécessaire afin de pouvoir stocker des données souvent nombreuses.
Une base de donnée est un outil permettant de stocker et accéder à des données.
On peut distinguer deux types de bases de données :

- Base de données relationnelles (organisation relationnelle, à l'aide d'un SGBD, chaque entrée sera organisée sur une ligne)
- Base de données orientées colonnes (les données sont stockées en colonne et non par ligne. Cela permet d'ajouter des colonnes plus facilement, mais aussi une compression par colonne)

Cassandra fait partie des bases de données orientées colonnes.

Créé par Facebook en 2007 pour des besoins internes, Cassandra est optimisée pour la Big Data, pour des infrastructures avec des millions de requêtes ou une infrastructure cloud. En 2009, Facebook a rendu le projet open-source, et appartient à la fondation Apache.
Aujourd'hui , cette base de données est utilisée notamment par Github, Instagram, Netflix ou encore Twitter.
Cassandra possède plusieurs équivalents à savoir Hbase, Apache, Accumulo.

Cassandra est basé sur deux concepts de bases de données 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

De plus la base de données fonctionne avec des clusters. Il repose donc avec un ensemble de noeuds qui vont communiquer entre eux. Seulement ici, les noeuds sont tous égaux. Il n'existe pas de noeud maître ou de processus centralisant leur gestion

Ses objectifs sont clairs :

- Gérer une grande quantité de données
- Résister aux pannes
- Être performant

et possède de nombreux 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

Théorème CAP :

Les théorème CAP établit les paramètres sur lesquels on peut jouer pour configurer une base de données distribuée

- Cohérence (C pour Consistency)
- Disponibilité (A pour Availability)
- Tolérance aux pannes et coupures réseaux (P pour Partition-tolerance)

Il postule sur le fait que l'on ne peut choisir que 2 paramètres sur 3 et jamais les trois. Ici on trouve les couples possibles. théoriquement :

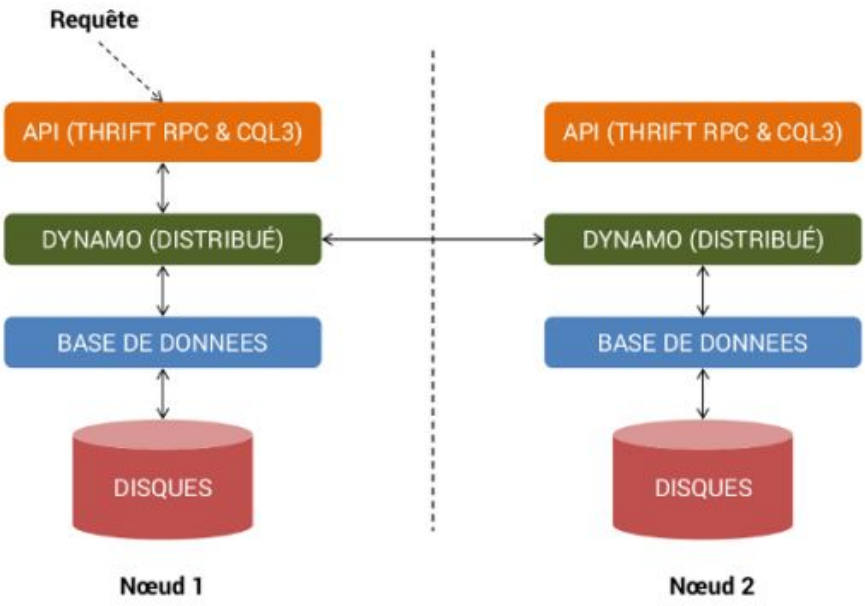
- CA (non résistante aux pannes)
- CP (non disponible à 100%)
- AP (non cohérente à 100%)

Or, dans la réalité, le P est plus ou moins imposé du fait que l'on n'est jamais à l'abris d'une panne réseau.

Architecture :

Cassandra est organisée en 3 couches. Elle s'inspire du papier de recherche Big Table de Google, ainsi que l'architecture Dynamo d'Amazon.

- 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

Ce qui différencie les deux modèles de données est sa capacité. D'un côté on va être limité à 2go, quand de l'autre côté on va être limité à 10 puissance 9. L'écart est géant. Mais ce ne sont pas ses seuls avantages :

- Le nom de la colonne est une valeur
- les colonnes sont triées par leur nom
- Possibilité de faire un lien entre deux entrées
- Possibilité que le nom de la colonne se suffise à lui seul, sans avoir forcément le couple nom + valeur. Cela permet d'optimiser les performances (Valueless column)
- Les lignes (rows) ont deux usages :
    - Wide Row : contient des milliers, millions de colonnes (ex: timeline twitter)
    - Skinny Row : en contient moins

Modèle Physique de données :

Keyspaces et Tables

Dans un cluster Cassandra on va retrouver :

- Des Tables
- Des Keyspaces (ressemble à une base)

Dans un modèle multi-tenant, on sépare les données de chaque partie dans des keyspaces. Dans ces keyspaces on va trouver les tables
Les tables dans une base de données comme Cassandra 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.

- Ces 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. Seulement, une partition doit toujours se trouver entièrement sur un seul noeud.

Il existe plusieurs choix pour le partitionnement au niveau du keyspace :

- BytesOrderedPartitioner
- RandomPartitioner (ou Murmur3Partitioner pour Cassandra V2)

Le BytesOrderedPartitioner, va partitionner les données de façon structurée et organisée. Chaque noeud prend en charge une plage de clé de partition triée par ordre croissant. Seulement, cela peut poser des problème d'équilibrage des noeuds

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
- ….

Le RandomPartitioner quant à lui va partitionner les données de manière aléatoire. Chaque noeud va prendre en charge une plage de la clé de partition distribuée uniformément. Il est donc conseillé d'utiliser le RandomPartitioner pour mieux répartir les charges.

Tokens

Les tokens sont un moyen simple de répartir les données dans le cluster. Il permet de distinguer les partitions et répartir uniformément un intervalle (plage) à chaque noeud.

Le Noeud Coordinateur

Comme nous l'avons évoqué plus haut, dans la base de donnée Cassandra, il n'existe pas de notion de noeud maître/esclave. Tous les noeuds sont égaux. Cela garantie l'absence de point unique de défaillance.

Le client fournit toujours une partition qui sera hashé. Si le hash de la partition ne correspond pas à sa plage, il va rediriger la requête vers un autre noeud, qui lui sera responsable de la donnée.

Ce système de noeud coordinateur permet la redirection de la requête au noeud concerné, et n'est pas unique à un seul noeud. Tous les noeuds peuvent jouer le rôle de coordinateur.

Structure d'une Table :

Les tables sont fortement typées

- Clé de partition (#partition)
- Clé de colonne (#col)
- Cellule

*Exemple*s :
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**

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
Pour résumer les méthodes API Thrifts :

- 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 réussi :

- 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 réplicas ont été persistées durablement
    - All : Je veux être certain que toutes les réplicas 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.
Chaque colonne possède un timestamp. Ce timestamp permet de faire fois pour savoir quelle donnée est la plus récente.

Performances en écriture :

En écriture, Cassandra est extrêmement rapide par :

- Son écriture dans le fichier Commit Log
- Son écriture en mémoire
- Pas d'update

Performance en lecture :

En lecture Cassandra est quasiment aussi performant :

- Il va contacter 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 Gossip est 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, toutes les secondes chaque noeud va échanger des messages avec 3 noeuds maximum. Ils vont échanger des données mais aussi leur état. 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, pour ensuite lui transmettre.

CQL 3

Pour manipuler les données avec la base de données Cassandra orientée colonne, cela fonctionne comme en SQL.
On va utiliser le CQL pour exécuter les requêtes et accéder aux données.

Exemples :

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

Contrairement au SQL, il n'existe pas de GROUP BY. Il faudra gérer les agrégations côté client

La clause WHERE est limitée. Possibilité d'égalité, inégalité, mais par contre, il est Impossible d'utiliser le *to_char*, *to_date* ou *like*
La clause ORDER BY est limitée. Il est possible d'utiliser un order by mais seulement sous certaines conditions
Il n'y a 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.

Sources :

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/#