Introduction à Cassandra

Houcine Senoussi

October 17, 2017

1 Introduction

2 Architecture

3 Modèle de données

4 Cohérence des données dans cassandra

5 Conclusion

6 Bibliographie

Introduction

Présentation de Cassandra :

- Cassandra a été conçue pour une manipulation performante de grandes masses de données structurées, semi-structurées et non structurées.
- C'est une base de données open source, non relationnelle (NoSQL), orientée colonne, distribuée, décentralisée, hautement scalable, permettant une grande disponibilité des données (highly available), tolérante aux pannes et dont la cohérence (consistency) est réglable.

Introduction

Présentation de Cassandra-Suite :

- Cassandra s'est inspirée de Dynamo (Amazon) pour son architecture et de Bigtable (Google) pour son modèle de données.
- Cassandra a été créée chez Facebook et est aujourd'hui utilisée notamment par eBay, Netflix, UBS et Coursera.

Cassandra et le théorème de Brewer

- Comme nous l'avons vu au dernier chapitre, cassandra appartient au coté *AP* du triangle représentant le théorème de Brewer :
  - grande disponibilité des données (A),
  - grande tolérance aux pannes (P),
  - en revanche, la cohérence des données (consistency (C)) n'est pas stricte. Elle est qualifiée de **faible**, de **finale** ou de **réglable**.

Architecture

- La première caractéristique de Cassandra est d'être **distribuée** : les données sont réparties sur plusieurs machines connectées (formant un **cluster**). La distribution des données est transparente à l'utilisateur.
  - En général, un cluster est constitué de plusieurs milliers de **noeuds** pouvant être répartis sur plusieurs data centers.
- Tous ces noeuds sont identiques, il n'y a ni maître ni escalve. On dit que Cassandra est **décentralisée** et qu'elle utilise une approche **peer-to-peer**.
  - Tous les noeuds peuvent accepter et gérer une demande de lecture ou d'écriture indépendamment de la localisation de la donnée concernée. Autrement dit, un client peut s'adresser à n'importe quel noeud et ce dernier jouera le rôle de proxy avec les noeuds contenant la donnée concernée.

Architecture

- Tous les noeuds sont deux-à-deux connectés. Ils communiquent en utilisant un protocole appelé '**Gossip**'.
  - Ce protocole permet aux noeuds d'échanger régulièrement des informations concernant leur état et celui d'autres noeuds dont ils ont connaissance.
  - Cela permet, entre autre, de détecter les noeuds qui ne sont pas utilisables (provisoirement) : en panne ou ayant des performances faibles. Cela permet aussi de détecter à l'inverse ceux qui sont ' de retour'.

Architecture

- Cassandra présente les noeuds d'un cluster sous forme d'un **anneau** (**Ring**) (1).
- Cette présentation sert essentiellement dans l'affectation des données aux noeuds (où faut-il écrire une donnée ? où faut-il aller la chercher lorsqu'on a besoin de la lire ?).

Architecture

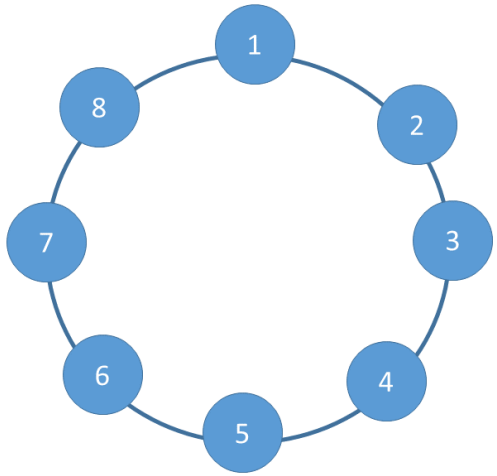

Figure: Présentation des noeuds dans un cluster

Architecture

- Les données sont **répliquées** : chaque donnée existe en plusieurs exemplaires réparties sur plusieurs noeuds.
  - Conséquence de l'approche peer-to-peer : toutes les copies ont la même valeur. Il n'y a pas de copie-maître qui fait autorité.
- Le placement de chaque donnée (c'est-à-dire de chaque copie) est effectué de la manière suivante :
  - Le nombre de copies de chaque donnée est déterminé au moment de la création de la 'base de données' (la keyspace, voir ci-dessous). Il s'agit du paramètre '**Replication factor**'.
  - La répartition se fait au niveau de 'lignes' : une 'table' est répartie sur plusieurs noeuds mais une ligne n'est pas découpée.

Architecture

- Le placement des données (suite) :
  - Chaque ligne possède une clé primaire (voir ci-dessous, modèle de données). Cette clé sert à calculer une valeur de hachage (hash value).
  - Cassandra associe à chaque noeud une plage de valeurs de hachage.
  - La première copie d'une ligne est donc affectée au noeud dont la plage contient la valeur de hachage.
  - Les copies suivantes sont affectées en fonction d'un paramètre appelée 'Stratégie de réplication' (**Replication strategy**).

Architecture

- Le placement des données (suite) :
  - Il existe deux stratégies de réplications :
    - **SimpleStrategy** : utilisable lorsqu'on a un seul datacenter. Une fois que la première copie est affecté à un noeud, les autres sont affectées aux noeuds suivants (sur le ring) dans le sens des aiguilles d'une montre.
    - **NetworkTopologyStrategy** : À utiliser lorsqu'on dispose de plusieurs datacenters. Elle précise combien de copies sont à placer dans chaque datacenter.

Architecture

- Le caractère décentralisé de Cassandra favorise les points forts suivants :
  - Une grande **tolérance aux pannes** : contrairement aux architectures maître-esclave il n'y a pas de point de passage obligatoire (le maître) dont une panne éventuelle perturbe fortement l'ensemble du réseau.
  - Une grande **disponibilité** des données : conséquence de la réplication mais aussi de l'absence d'une copie maître : si une copie n'est pas disponible on accède aux autres.

Architecture

- Le caractère décentralisé de Cassandra favorise les points forts suivants (suite):
  - Une grande **scalabilité** : la possibilité d'ajouter un nouveau noeud sans rien modifier à la structure.
    - Cette scalabilité est qualifiée d'**élastique** : la possibilité d'ajouter des noeuds (scale up) puis d'en supprimer (scale down).
    - Elle est aussi qualifiée de **linéaire** : la performance croît linéairement en fonction du nombre de noeuds.

Modèle de données

- En cassandra, Le conteneur principal qui contient l'ensemble des données d'une application (gestion commerciale, gestion de la scolarité) s'appelle une **keyspace**. C'est l'équivalent d'une base de données relationnelle.
- Une keyspace est composée de **tables**. Dans les premières versions de cassandra, on parlait plutôt de 'famille de colonnes' (**Column Family**).
- Une table possède obligatoirement une clé primaire.
- Cassandra, dans ses versions récentes, reprend donc les bases du modèle relationnel, mais à la différence de ce dernier, Cassandra a des tables dénormalisées (par exemple : il n'y a ni clé étrangère ni jointure).
- Cassandra dispose d'un langage de requêtes (CQL pour Cassandra Query language) dont la syntaxe est largement

Types de données

- Nous avons les types habituels (numériques, caractères et chaînes de caractères, date, ...).
- Nous avons les types **uuid** et **timeuuid** servant à générer des identifiants unique :
  - une variable uuid est une suite de 32 chiffres hexa. On peut les générer à l'aide dela fonction **uuid()**.
  - Le type timeuuid permet aussi de générer des identifiants unique en utilisant l'heure. On peut générer ces valeurs avec la fonction **now()**.
    - On peut récupérer ce type de variables sous la forme d'une date en utilsant la fonction **dateof()**.

Types de données

- Nous avons le type **timestamp** dont les valeurs sont des entiers longs (64 bits) représentant le nombre de millisecondes écoulées depuis une date fixe(1/1/1970).
  - Une telle valeur peut être donnée sous la forme d'une chaîne de caractères pouvant prendre plusieurs formats (exemple 'yyyy-mm-dd' ou 'yyyy-mm-dd HH:mm:ss') ou par la chaîne 'now'.
- Nous avons le type **counter** (compteur) dont les valeurs sont des entiers longs (64 bits). La caractériqtiques principale de ce type est qu'il ne supporte que deux types d'opérations : incrémentation et décrémentation. Conséquence : il ne peut intervenir dans une requête insert mais uniquement dans les requêtes update.

Types de données

- Dans une table cassandra, les données ne sont pas toujours atomiques.
  - Nous disposons des types **set**, **list**, **map** et **tuple** pour mettre plusieurs valeurs dans une même colonne.

Types de données

- Un utilisateur peut créer ses **propres types**.
- Une ligne peut être insérée dans une table pour une durée limitée à l'issue de lequelle elle sera effacée. Ceci est possible grâce à l'option **TTL** = nombre-de-secondes.
- Dans une table Cassandra une colonne peut être statique : sa valeur est commune pour toutes les lignes.
- Une suite de requêtes DML peut être incluse dans un batch pour une exécution tout-ou-rien.
- Une requête insert peut inclure une option '**IF NO EXISTS**'.

Cassandra-Cohérence réglable

- La cohérence (Consistency) signifie que la lecture d'une donnée retourne toujours la dernière valeur écrite. Ce qui implique que plusieurs clients qui lisent cette donnée en même temps auront la même valeur.
- Cassandra "sacrifie" partiellement sa cohérence pour obtenir une disponibilité ("availability") totale.
- Certains qualifient cassandra de "eventually consistent" (cohérence finale). Ses concepteurs parlent de de "tuneable consistency" (cohérence réglable) : cassandra vous permet de décider du niveau de cohérence que vous voulez (ce qui influera en conséquence sur le niveau de disponibilité).

Cassandra-Cohérence réglable

- Le choix du niveau de cohérence peut être fait globalement (au niveau du cluster) ou au niveau de chaque opération de lecture ou d'écriture (selon les besoins de l'application).
- Opération de lecture : fixer le nombre de copies devant avoir la même valeur pour donner une réponse à l'application. Si ce nombre n'est pas atteint cassandra lance une opération de mise à jour des copies de cette donnée.
- Opération d'écriture : fixer le nombre de copies devant être mise à jour pour considérer que l'écriture a réussi.
- On considère qu'une cohérence est forte lorsque nous avons :
  - $R+W > Replication\_factor$, où R est le niveau de cohérence de la lecture et W celui de l'écriture.
  - Exemple : $R=W=2$ et $Replication\_factor=3$.

Conclusion

- Cassandra : Caractéristiques et fonctionnement.

Conclusion

- Managing Big data : SQL or NoSQL ? IDC 2012.
- Documentation Cassandra 3.0. Datastax.2016.