CONSERVATOIRE NATIONAL DES ARTS ET METIERS

Centre régional du Grand Est

MÉMOIRE

présenté en vue d'obtenir l'unité d'enseignement

ENG221 - Information et communication pour l'ingénieur - Oral probatoire

Dans le cadre du DIPLÔME D'INGÉNIEUR CNAM

SPÉCIALITÉ : INFORMATIQUE

OPTION : IRSM

par

Jérémy SNIDARO

Bases de données NoSQL : le cas de Cassandra

Soutenu le 15 juin 2021

JURY

MEMBRES : Monsieur Olivier FLAUZAC

Monsieur Benoit MULLER

Madame Safia SIDE

Introduction

Le monde des bases de données a longtemps été dominé par le modèle relationnel (SQL) basé sur des données structurées. Avec l'avènement du Web et des grandes entreprises collectant des quantités énormes de données, l'utilisation du modèle relationnel est arrivée à ses limites, des nouveaux besoins ont émergé. On parle alors de schéma non structuré, de haute disponibilité, de réplication et de scalabilité.

En réponse à ses nouveaux besoins, de grande entreprise comme Google, Facebook ou bien Amazon ont créé leurs propres systèmes de gestion de base de données. Ces systèmes ont mené à la création du modèle non relationnel NoSQL.

Le NoSQL apporte beaucoup de changements dans le domaine des bases de données. Pour mieux cerner les cas d'utilisation et certaines de ces fonctionnalités clés nous allons nous intéresser à la base de données NoSQL Open Source Cassandra.

Apache Cassandra est une base de données NoSQL décentralisée et distribuée, hautement disponible, linéairement extensible, tolérante aux pannes, elle excelle particulièrement dans les gros volumes de données réparties sur plusieurs datacenters et nécessitant un haut niveau de disponibilité. Créer au sein de Facebook elle combine les fonctionnalités de deux bases de données NoSQL, le stockage distribué et la réplication de DynamoDB d'Amazon et le moteur de Bigtable de Google, Cassandra est aujourd'hui utilisée par beaucoup de grosse entreprise comme Netflix, BlackRock et Apple [1].

Bien que Cassandra partage certains concepts des bases de données relationnelles, comme le fait de supporter les schémas de données structurées, de stocker ses tables sous forme de lignes et de colonnes, et d'utiliser un langage se rapprochant fortement du SQL pour effectuer ses requêtes. Il ne faut pas oublier qu'elle reste tout de même une base NoSQL qui répond à des besoins différents et elle ne doit pas être utilisée comme une base de données à tout faire.

Ce travail effectué au sein de la formation au diplôme d'ingénieur du CNAM a pour objectif, de présenter les fonctionnalités et les cas d'utilisation de la base de données NoSQL Cassandra, ainsi que de développer et d'évaluer mes capacités à synthétiser l'information technique et à la présenter devant un public professionnel avertit.

Nous verrons dans un premier temps les concepts généraux et les besoins auxquels réponde le modèle non relationnel ainsi que les bases de données NoSQL les plus utilisées en abordant plus particulièrement les avantages et les inconvénients de ce modèle, puis nous détailleront le fonctionnement et ce qu'apporte Cassandra dans le monde des bases de données.

I - NoSQL

Le NoSQL pour “Not only” SQL est apparu dans les années 2000 [2] en réponse au nouveau besoin du web. C’est un modèle non relationnel qui permet d’être plus flexible au niveau de la structure des données, et de mieux gérer des gros volumes de données.

L’intérêt grandissant pour les bases de données NoSQL reflète les nouveaux besoins du Web et du Cloud, aujourd’hui de plus en plus d’entreprises ont besoin de solution scalable, pouvant gérer des gros volumes de données, flexible, hautement disponible et tolérante aux pannes.

I.1 - Base de données

Il existe une multitude de bases de données NoSQL utilisant des types de stockages différents. Les plus courantes sont clés-valeur, documents, graphes et orientées colonne.

Chaque base de données essaye de répondre à des besoins différents, par exemple parmi les plus utilisées nous avons :

- MongoDB (Document)
- DynamoDB (Clé-valeur et Document)
- Neo4j (Graphe)
- Cassandra (Orienté Colonne)

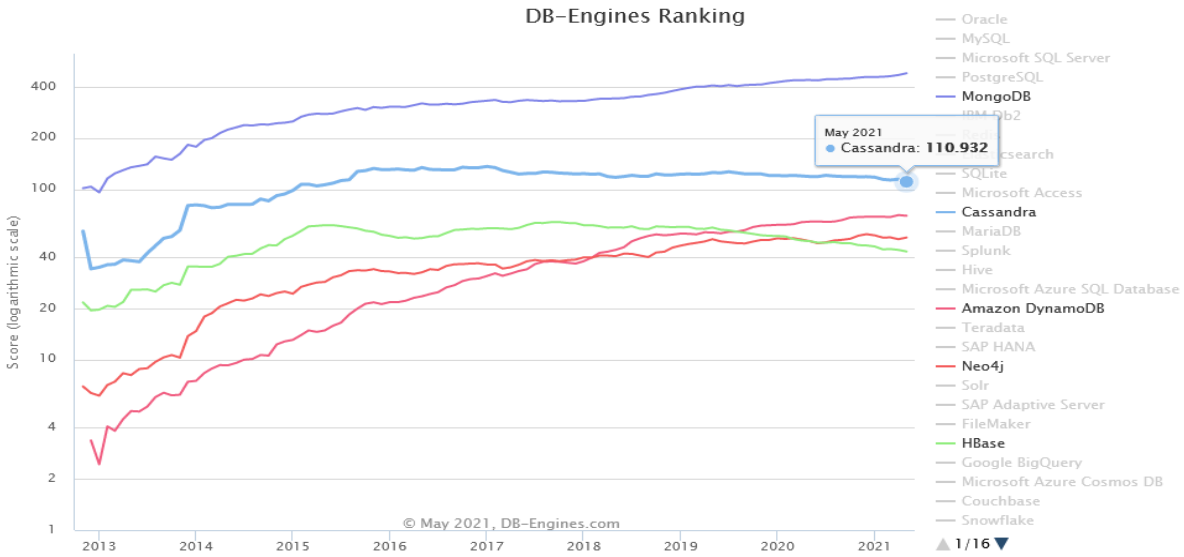

Figure 1 Popularité de Cassandra comparé à d’autres bases NoSQL, d'après le site db-engines.com [22]

I.2 - Type de stockage

- Document

L'unité de stockage dans la base de données est le document, il est généralement représenté dans le format JSON ou bien XML, les requêtes sont la plupart du temps effectuées via une API.

L'on peut directement requêter des champs contenus dans le document :

2Json Document inside DynamoDB

Par exemple avec l'API de DynamoDB on peut extraire le champ « Author » contenu dans le JSON à l'intérieur du champ information avec l'appel suivant :

```
response = table.get_item(Key = {'subject': "NoSQL Cassandra"}, ProjectionExpression = "information.Author")
```

```
Response
{
  "information": {
    "Author": "SNIDARO Jérémy"
  }
}
```

- Clé-valeur

Les systèmes clés-valeur considère chaque entrée dans une table de manière unique, chaque entrée peut posséder des champs différents des autres. Cela apporte une grande flexibilité.

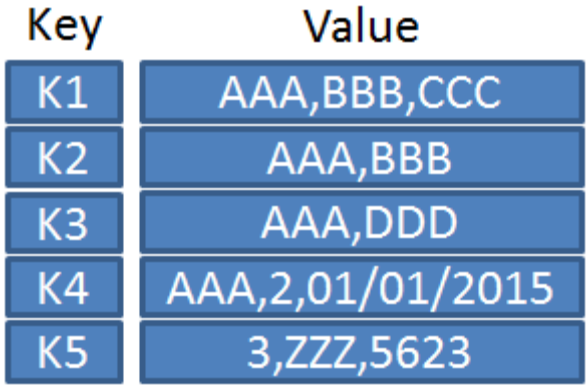

Figure 3 Exemple clé-valeur avec un nombre de colonnes dynamiques [23]

Le stockage sous forme de clé-valeur est propice aux applications nécessitant de gros volumes de données non complexes ou ayant une fréquence de modification élevée, tel que des valeurs de marché financier, ou bien encore des données de capteurs [3 p115].

Les cas d'utilisation les plus courants sont, la gestion du cache, file de message et la gestion de session.

- Graphe

Certaines applications ont besoin d'un lien fort entre les données, ce type d'application tirera parti du schéma de données graphe ou chaque entité (donnée) peux être reliée à une autre par une relation.

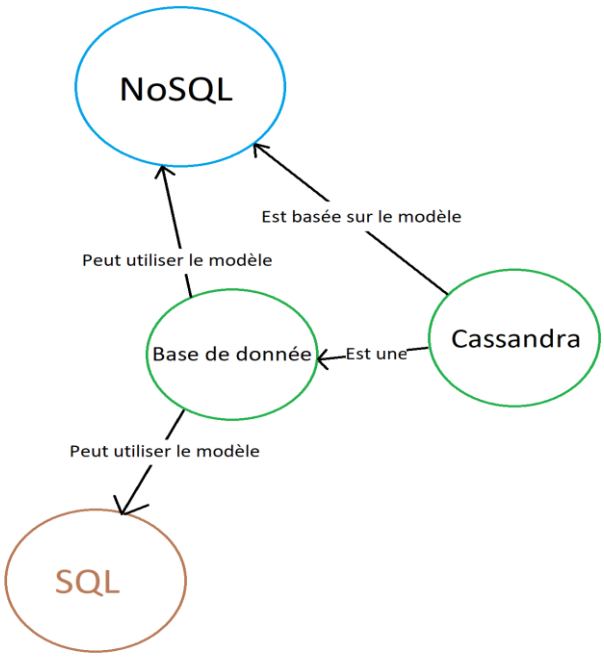

Figure 4 Exemple relation base de données graphe

- Colonne

Les bases de données NoSQL utilisant un stockage de type colonnes sont celles qui se rapprochent le plus des bases de données classiques (SQL) qui stockent ses données par ligne.

Prenons comme exemple la table suivante :

id	Jury_name	School	Genre
1	SIDE	CNAM	F
2	FLAUZAC	CNAM	M
3	MULLER	CNAM	M

Figure 5 Exemple de table d'une base de données

Dans une base SQL les données sont stockées sous cette forme :

1,SIDE,CNAM,F;1,FLAUZAC,CNAM,M;3,MULLER,CNAM,M

Dans une base de données NoSQL elles seront représentées sous la forme suivante :

1,2,3;SIDE,FLAUZEC,MULLER ;CNAM,CNAM,CNAM;F,M,M

Le principe est d'utiliser une table et de stocker les données sous forme de colonnes. Les différences principales avec le stockage par ligne du SQL sont :

1 – Le nombre de colonnes par entrée n'est pas statique, deux entrées dans une table peuvent posséder un nombre de colonnes différentes, les valeurs Nulles ne sont pas stockées.

2 – Certaines requêtes pour de l'analyse ou de la statistique sont bien plus rapides, au lieu de lire les données lignes par lignes pour prendre la colonne qui nous intéresse on peut simplement sélectionner toutes les données de la colonne souhaitée.

3 – Le stockage de valeur identique à la suite n'est pas nécessaire, la base de données peut spécifier que les x valeurs suivantes seront les mêmes et ainsi ne pas avoir à stocker plusieurs fois la même valeur, dans notre exemple ci-dessus, la base NoSQL ne stockera qu'une fois la valeur « CNAM ».

I.3 - Propriété et Concept généraux

Les bases NoSQL possèdent différentes propriétés qui caractérisent comment les données sont traitées et quelles sont les garanties associées aux traitements des données.

Ce chapitre met en avant les propriétés et les concepts que les bases NoSQL partagent entre elles.

I.3.1 - Théorème CAP

Le théorème CAP [4] ou CDP en français, définit qu'il est impossible pour un système informatique de satisfaire les trois contraintes suivantes en même temps :

- Cohérence, la capacité à un système à avoir à plusieurs endroits exactement les mêmes données au même moment
- Disponibilité, toute requête doit recevoir une réponse
- Tolérance aux partitionnements/pannes, en cas de panne d'une partie du système ledit système doit continuer de fonctionner de manière autonome

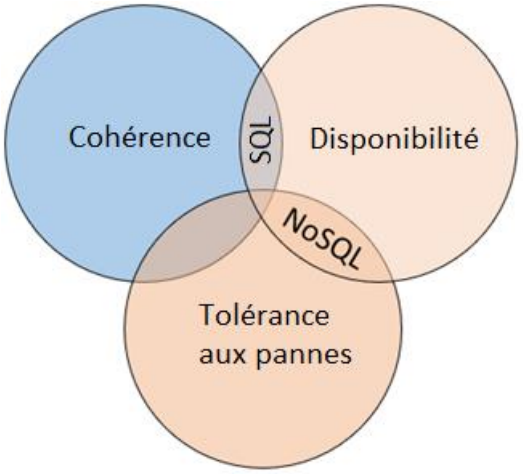

Figure 6 Théorème CAP

NoSQL ne fait pas exception à ce théorème et répond à la contrainte de la disponibilité et de la tolérance aux pannes. Ne pouvant pas garantir la cohérence totale des données, le concept de cohérence éventuelle est appliqué.

I.3.1.1 - Disponibilité

La disponibilité est la capacité à un système à répondre aux requêtes même en cas de défaillance d'une partie du système. Pour répondre à cette propriété, NoSQL utilise un modèle où les données sont répliquées sur des nœuds indépendants les uns des autres, ces nœuds peuvent aussi être répartis dans différents datacenters et à des localisations géographiques différentes. En contrepartie de cette haute disponibilité, la réplication des données entre les nœuds n'est pas assurée et peut subir un délai, c'est pourquoi la Cohérence des données n'est pas garantie, on parle alors de Cohérence éventuelle.

I.3.1.2 - Cohérence éventuelle

L'éventuelle cohérence des données signifie que si une donnée est modifiée et lue immédiatement après, il n'est pas garanti que la modification de la donnée soit prise en compte à tous les endroits où la donnée est stockée, la lecture peut se faire sur la version précédente de la donnée. Cela est dû aux faites qu'il y a un délai de propagation de la modification entre tous les nœuds.

I.3.1.3 - Tolérance aux pannes

Un système tolérant aux pannes a la faculté de détecter les pannes et de se remettre dans son état initial et ainsi il lui est possible de continuer de fonctionner et de maintenir ses propriétés.

NoSQL assure la tolérance aux pannes par la redondance des données dans différents nœuds. Au lieu de ne stocker qu'une seule fois la donnée, elle est répliquée dans plusieurs nœuds.

I.4 - Transaction

Une transaction représente un changement effectué sur une base de données, cela peut être l'ajout d'une entrée, ou bien encore une modification dans une table.

La transaction a trois fonctions :

- Assuré la restauration en cas d'échec ou d'interruption de la transaction.

- Isoler les différentes demandes d'accès à la base de données au même moment, en empêchant par exemple deux écritures simultanées sur la même donnée.

- Permettre de regrouper plusieurs changements dans une même requête et l'annuler en cas d'erreur sur un ou plusieurs éléments de la requête.

De manière générale le concept de transaction n'est pas utilisé dans les bases de données NoSQL, cependant certaine base de données le supporte.

I.5 - Avantage

- Supporte de gros volumes de données

NoSQL étant extensible linéairement il suffit d'ajouter des serveurs pour supporter plus de requêtes et augmenter l'espace disque total disponible.

- Schéma de donnée flexible

Chaque entrée dans la base de données étant traitée de manière individuelle, le schéma de données peut être modifié au cas par cas pour chaque entrée sans affecter les autres, rajouter une colonne en SQL est très couteux par exemple.

- Supporte tout type de schéma de données

Il y a une multitude de bases de données NoSQL qui répond à des besoins spécifiques et supporte des schémas de données différents.

- Haute disponibilité et tolérance aux pannes

La décentralisation permet une de maintenant une haute disponibilité en ayant des systèmes indépendants les uns des autres travaillants ensemble. La distribution des données signifie que les données sont répliquées à plusieurs endroits, cette caractéristique assure la tolérance aux pannes.

- Coût faible

Atteindre le même niveau de disponibilité, de tolérance aux pannes, de performance est quelques choses de très couteux en SQL. NoSQL par sa conception et sa capacité à s'étendre

de manière linéaire pour correspondre aux volumes des données permet d'avoir un coût de fonctionnement plus faible.

- Performance

Les requêtes en écriture sont très rapides et celle en lecture qui ne nécessite pas une grande complexité le sont aussi.

I.6 - Inconvénient

- L'intégrité des données n'est pas garantie à 100%

La cohérence étant éventuelle, certain cas d'utilisation sont à proscrire, comme des opérations bancaires.

- Langage de requête non normalisé

Chaque base de données possédant son propre langage ou API, une migration d'une solution à une autre devra passer par la réécriture complète des requêtes

- Non adapté aux requêtes complexe ou dynamique

Le concept de jointure entre tables n'existe pas, il faut penser chaque table en y incluant les données que l'on souhaite, peu de système de base de données supporte la transaction.

NoSQL apporte de nouvelles fonctionnalités et comble les manques des bases de données relationnelles, il n'a pas vocation à remplacer le SQL. Grâce à la diversité des bases de données NoSQL disponible les utilisateurs ont la possibilité de choisir la base de données la plus adapté à leurs besoins. Dans le chapitre suivant nous allons nous pencher sur le cas de la base de données Cassandra.

II - Apache Cassandra

Dans le chapitre précédent nous avons abordé comment NoSQL répond aux nouveaux besoins du Web et du Cloud, ainsi que les concepts généraux et les fonctionnalités qui en découlent. Dans ce chapitre nous allons aborder l'implémentation du modèle NoSQL au sein de la base de données Cassandra.

II.1 - Définition

Cassandra est l'une base de données NoSQL les plus utilisés, elle est open source, codé en java, orienté colonne, décentralisée et distribuée, hautement disponible, linéairement extensible et tolérante aux pannes. Elle excelle particulièrement dans les gros volumes de données réparties sur plusieurs data centers et nécessitant un haut niveau de disponibilité. Créer au sein de Facebook puis repris par la fondation Apache en 2009 [5], elle combine les fonctionnalités de deux bases de données NoSQL, le stockage distribué et la réplication de DynamoDB d'Amazon et le moteur de Bigtable de Google. Cassandra est aujourd'hui utilisée par beaucoup de grosse entreprise comme Netflix, BlackRock et Apple [1].

Nous parlerons des fonctionnalités de Cassandra à partir de la dernière version stable disponible au moment de l'écriture de ce mémoire, la version 3.11 [6].

II.2 - Objectif de Cassandra

Cassandra a été créer pour résoudre les problèmes liés à la gestion de gros volumes de données, elle fournit un haut niveau de performance sur les requêtes en lecture et écriture et fournis un haut taux de disponibilité et est extrêmement tolérante aux pannes.

II.3 – Nœud

Chaque serveur utilisant Cassandra possède un certain nombre de nœuds, les nœuds sont regroupés entre eux au sein d'un cluster. Les clusters peuvent être divisés en fonction de leur séparation logique ou physique. Chaque groupe de nœuds qui partage la même configuration informe Cassandra du nombre de copies d'une donnée à garder. Ils sont tous identiques et possèdent les mêmes fonctions. L'état de chaque nœud est partagé en utilisant le protocole Gossip [7].

Les nœuds sont répartis dans des racks, un rack au sein de Cassandra correspond à un groupe de serveur dans lequel chaque donnée n'est présente qu'une fois, cela permet de séparer les données répliquées sur différents racks éviter la perte de données en cas de perte d'un rack.

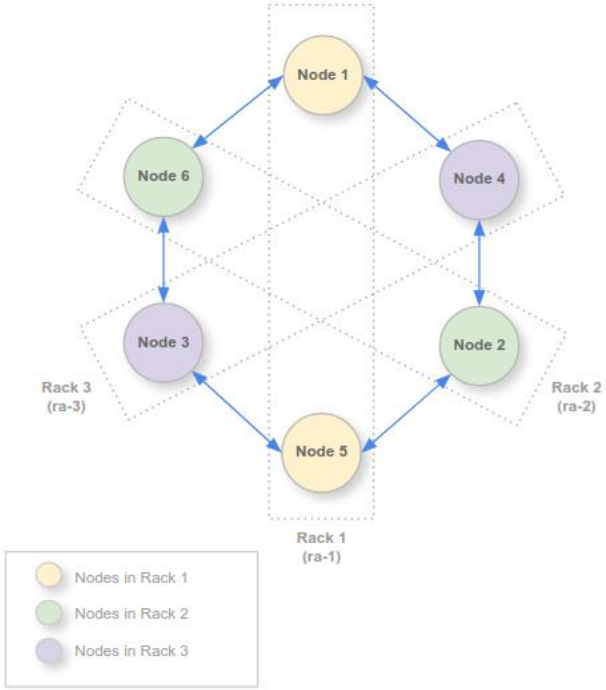

Figure 7 Représentation nœuds et racks

II.4 - Fonctionnalités de base

II.4.1 - Réplication

Grâce à ses multiples copies des données réparties sur plusieurs nœuds, Cassandra permet aux clusters de continuer de fonctionner même en cas de perte d'un ou plusieurs nœuds. Cela permet d'effectuer des maintenances et des mises à jour sur la base de données sans interruption de service.

II.4.2 - Extensibilité linéaire

Pour pallier la montée en charge des serveurs, Cassandra bénéficie d'une extensibilité linéaire, en ajoutant de nouveau serveur, la puissance disponible augmentera de manière linéaire sans limite.

II.4.3 - Cohérence configurable

La cohérence des données est configurable, pour chaque requête en écriture ou en lecture, il est possible de choisir le niveau de cohérence des données utilisée. Le niveau de cohérence souhaité influe sur la performance des requêtes.

II.4.4 - Performance

En plus de fournir une cohérence configurable, Cassandra possède une structure de données qui permet de mettre les données ayant une relation entre elles le plus proche possible les unes des autres. Cela permet de stockées les données qui nous intéresse dans le même nœud et dans l'ordre dans lequel on souhaite les rechercher.

Étant donné que Cassandra est une base de données distribuée, les performances sont améliorées en regroupant les données en partition au sein de nœuds.

Attention, il ne faut pas mettre un système de « load balancer » qui enverrait les requêtes à Cassandra, un tel système existe déjà au sein de Cassandra et cela provoquerait des interférences.

II.5 - Structure des données

Cassandra utilise un schéma de stockage des données orienté colonne avec un système ligne et de partition. Le schéma est la définition de comment les données sont disposées dans les familles de colonnes, une famille de colonnes se rapproche du concept de table en SQL.

La structure des données consiste en un Cluster, un Keyspace, des familles de colonnes (tables), des lignes, et des colonnes [8].

- Le cluster contient des Keyspaces situé sur un ou plusieurs nœuds
- Un Keyspace correspond à la base de données, il contient des familles de colonnes ainsi que la configuration de la base, la fonction du Keyspace est de permettre de configurer la réplication entre certaines tables.

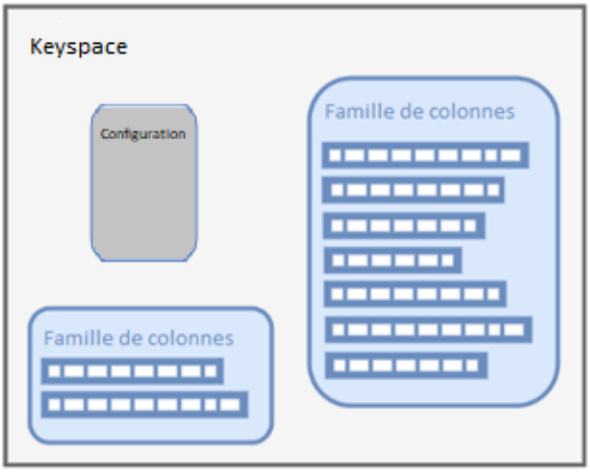

Figure 8 Keyspace et famille de clonnes [24]

- Une famille de colonnes contient des lignes, on peut faire l'analogie avec le concept de table en SQL

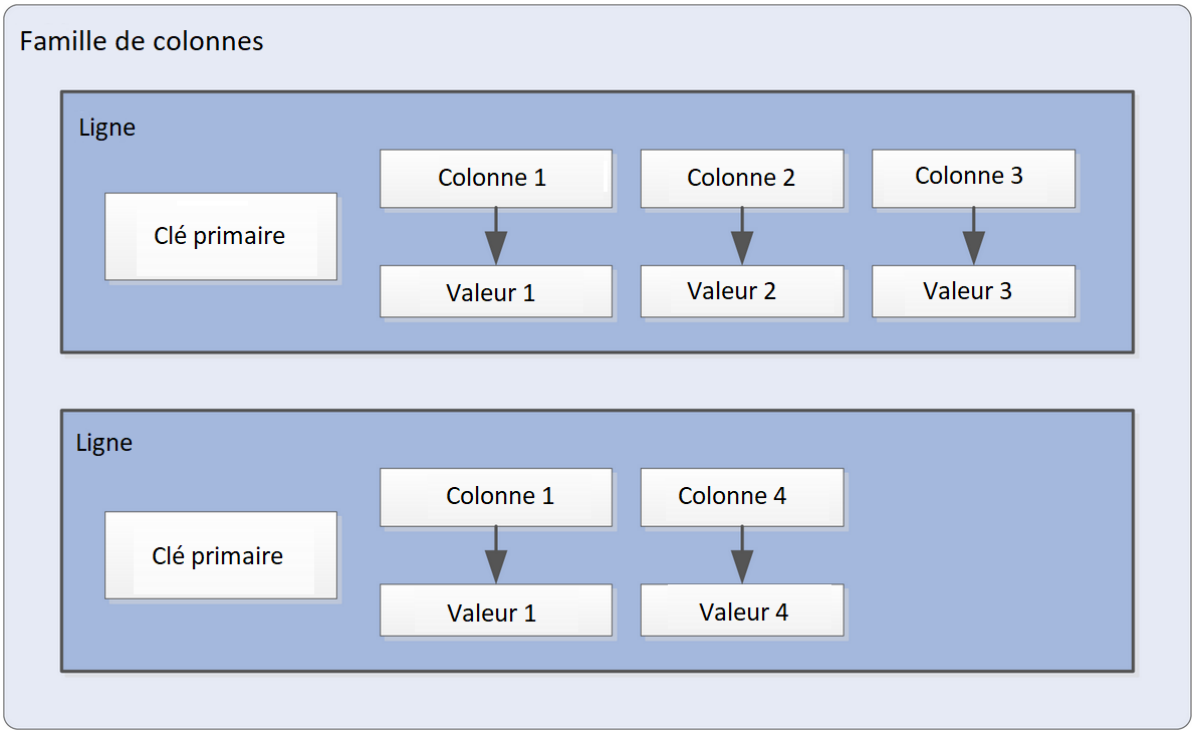

Figure 9 Représentation d'une famille de colonnes [13]

- Chaque ligne est identifiée par une clé primaire, elles peuvent contenir un nombre de colonnes différentes, la valeur du nom de la colonne peut être binaire, par exemple du texte, un nombre, une date, ect.
- Une colonne est l'association d'un nom et d'une valeur

II.5.1 - Famille de colonnes

Une famille de colonnes ou table est composée d'une suite de lignes ordonnées. Une ligne contient une liste de colonnes ordonnée et une clé primaire. Les lignes sont stockées dans des containeurs appelés partition.

La clé primaire identifie de manière unique une ligne, elle est obligatoire, elle correspond à une ou plusieurs colonnes.

Le concept de famille de colonnes permet de regrouper des données ayant une relation entre elles, de les organiser, trier, et les isoler entre elles dans le but d'augmenter les performances.

II.5.2 – Partition

Une partition est un regroupement de ligne qui partage la même valeur de clé de partition. Une ligne possède toujours une clé de partition, s'il n'y a pas de colonne du type clustering, la clé primaire est utilisée comme clé de partition.

Les lignes possédant la même partition ont là garanti d'être sur les mêmes nœuds.

Les lignes sont organisées en partitions et assignées aux nœuds dans le cluster de Cassandra. L'ordre des données au sein d'une partition est défini par la colonne clustering.

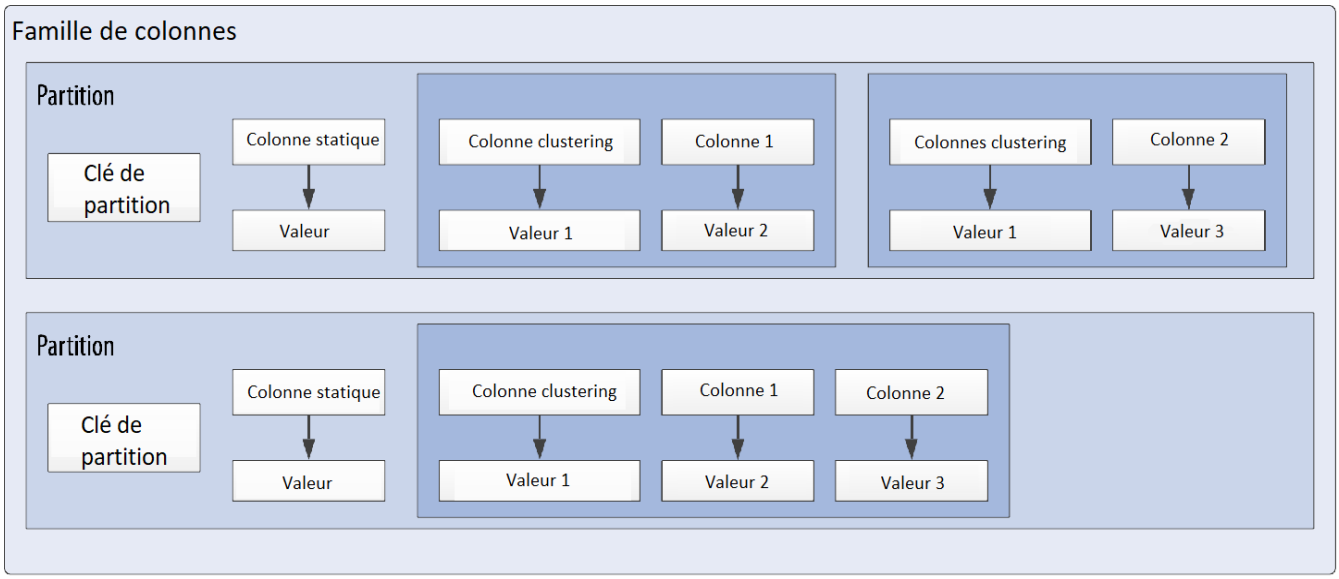

Figure 10 Famille de colonnes avec partition [13]

II.5.3 - Colonne Clustering

La colonne Clustering est une colonne qui est utilisée pour former la clé primaire, définir une colonne comme Clustering permet de définir comment les colonnes sont ordonnées au sein d'une ligne.

En définissant une colonne comme clustering, la ligne ordonnera les données en fonction de la valeur de la clé de partition puis de cette colonne.

Dans la figure suivante, nous voyons que les données sont triées par la colonne clé de partition en premier, ici « author_id » et ensuite par la colonne clustering, correspondant à « book_id » :

author_id	birthday	firstname	book_id	book_date	book_title	book_price
aaa	1/1/1970	alice	1	12/12/2014	AAA	10
aaa	1/1/1970	alice	2	17/12/2014	BBB	20
aaa	1/1/1970	alice	3	19/12/2014	CCC	30
bbb	1/1/1984	bob	1	13/12/2014	DDD	40
bbb	1/1/1984	bob	2	18/12/2014	EEE	50

Figure 11 Exemple triage avec clé partition et colonne clustering [25]

II.5.4 - Clé partition

Chaque partition est identifiée de manière unique par une clé de partition, la colonne clustering est utilisée pour identifier les lignes dans une partition. La partition permet de regrouper plusieurs lignes. Si aucune colonne clustering n'est présente chaque partition ne contiendra qu'une seule ligne.

II.5.5 - Colonne statique

Une colonne peut être déclarée statique, une colonne statique est partagée par toutes les lignes se trouvant dans la même partition. Une colonne définie comme clé primaire ne peut pas être statique.

II.6 - Modélisation des données

Pour tirer le meilleur parti de Cassandra il faut appliquer une modélisation des données cohérentes à notre utilisation. Ce processus consiste à identifier les données et les relations entre elles et à déterminer comment les données seront lues et quels types de requêtes seront les plus fréquentes. Ses informations permettent de schématiser et de construire les familles de colonnes de manière optimale.

II.6.1 – Requête et langage CQL

CQL pour « Cassandre Query Langage » [9, p289] est un langage permettant d'effectuer des requêtes et d'interagir avec la base de données, il ressemble au SQL, SQL peut s'utiliser avec CQL Shell (cqlsh) ou bien directement via un langage de programmation avec un drivers,

Les requêtes sont construites pour n'accéder qu'à une seule famille de colonnes (table) à la fois, toutes les données souhaitées doivent donc être dans la même famille de colonnes. Cette approche permet d'avoir des requêtes plus rapides.

Effectuer une requête sélectionnant toutes les données d'une famille de colonnes sans filtre (clause where) forcera Cassandra à récupérer toutes les données sur tous les nœuds, une requête doit toujours avoir un filtre sur la/les colonne(s) constituant la clé de partition ou sur une colonne indexée.

II.6.2 - Index

Cassandra permet d'indexer une ou plusieurs colonnes, l'indexation permet d'effectuer des requêtes avec comme filtre une colonne ne faisant pas partie de la clé de partition, cependant en fonction de la colonne indexée, l'indexation peut sérieusement diminuer les performances. La colonne index est cachée et séparée de la famille de colonnes dont provient la/les colonne(s) indexée(s).

L'utilisation d'index est plus efficace sur une famille de colonnes contenant beaucoup d'itération de la donnée. Il faut éviter d'utiliser les indexes sur les familles de colonnes subissant fréquemment des modifications.

II.6.3 - Token

La distribution des données se fait via l'utilisation de token. Un token est la valeur obtenue après l'utilisation d'une fonction de hachage sur la clé de partition de la requête [9, p253]. Quand un nœud est créé il lui est automatiquement ou manuellement alloué une plage de token. Par la suite quand des données doivent être ajoutées, la plage de token est utilisée pour savoir sur quels nœuds la donnée sera sauvegardée.

En interne un token est représenté par chiffre codé sur 64 bits.

En reprenant l'exemple de la table avec le jury et en ayant les noms des membres comme clé de partition, la répartition des membres du jury au sein d'un cluster Cassandra pourrait ressembler à la figure suivante :

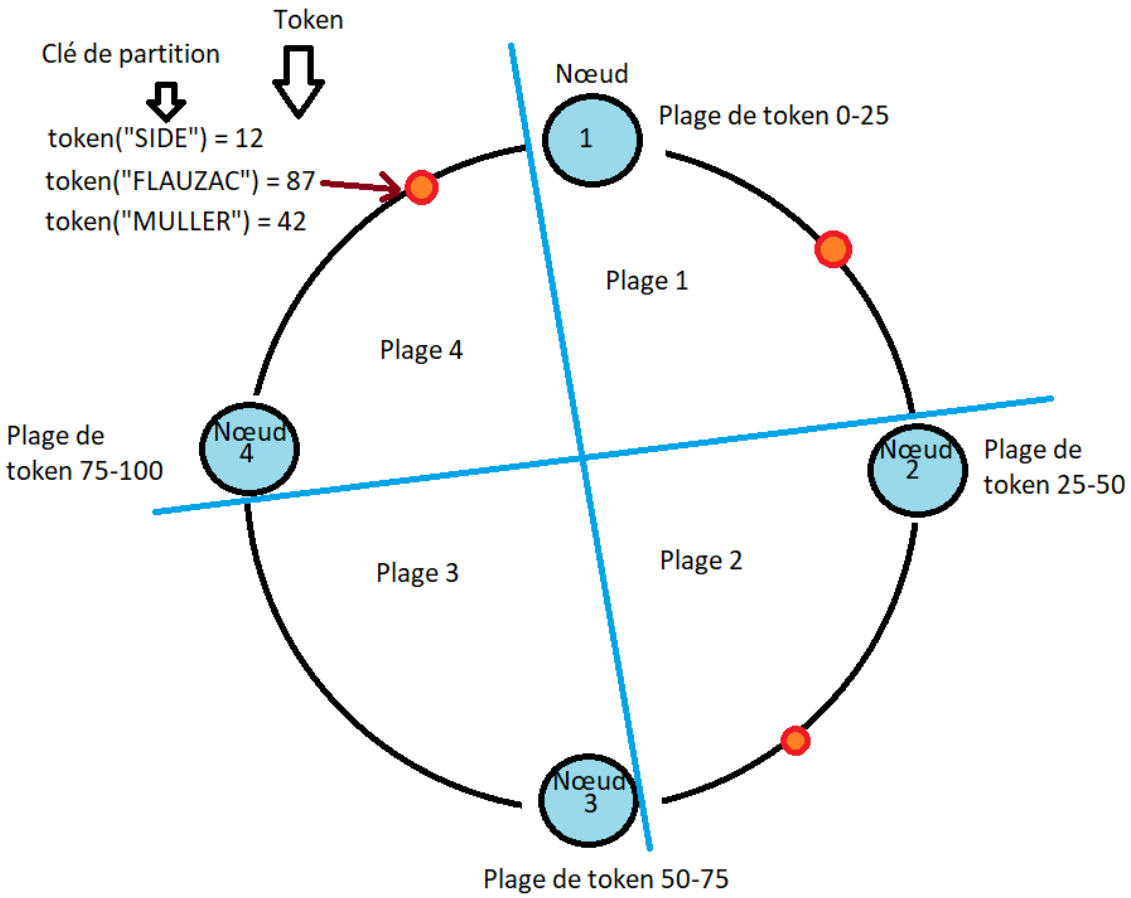

Figure 12 Token et plages de token associes aux nœuds

L'écriture des données et la structuration de celle-ci par Cassandra est complexe, mais elle permet d'optimiser le parcours des données. En effet grâce à la colonne clustering on peut définir l'ordre dans lequel les données seront présentes dans une ligne, sachant qu'une requête venant lire dans une ligne doit commencer à la lire depuis le début.

Mettre les données les plus lues au début, et les grouper avec d'autres colonnes souvent lues en même temps permet d'augmenter significativement la vitesse de lecture. L'utilisation de plage de token attribué aux nœuds permet de répartir les données au sein du cluster.

II.7 – Architecture

L'architecture de Cassandra a été pensée pour répartir la charge sur plusieurs nœuds interconnectés avec un système pair à pair, les données sont réparties sur les nœuds au sein d'un cluster. La communication entre les deux se fait via le protocole Gossip [10].

II.8 - Snitch

Un Snitch détermine le rack auquel un nœud doit appartenir [11]. Il permet à Cassandra d'avoir une vue d'ensemble de la topologie du réseau de nœud et peut ainsi rediriger les requêtes de manière optimale.

Grâce à cette topologie Cassandra peut répartir les données et assurer une réplication des données entre les différents racks et ainsi éviter d'avoir plusieurs copies d'une donnée sur le même rack.

II.9 - Facteur de réplication

Un cluster peut être configuré avec un facteur de réplication, il permet de déterminer le nombre de copies d'une donnée que le cluster doit posséder.

Le nombre de nœud possédant la copie d'une donnée est égal au facteur de réplication.

II.10 - Écriture des données

L'écriture des données se fait en trois étapes [12] :

- Enregistrement de l'écriture dans le commitlog
- Écriture des données dans la memtables
- Transfère des données de la memtables vers le disque dans un fichier SSTables

L'écriture sur les nœuds se fait de manière séquentielle en utilisant un système de commit log pour s'assurer de l'intégrité des données. Les données sont ensuite écrites dans une structure en mémoire appelé memtable, lorsque la structure est entièrement remplie, les données sont transférées sur le disque dans un fichier SSTables. Toutes les données sont automatiquement partitionnées et répliquées à travers le cluster. Quand une donnée doit être supprimée elle est tout d'abord marquée comme tombstone, lors du processus de compactage les données ainsi marquées sont supprimées.

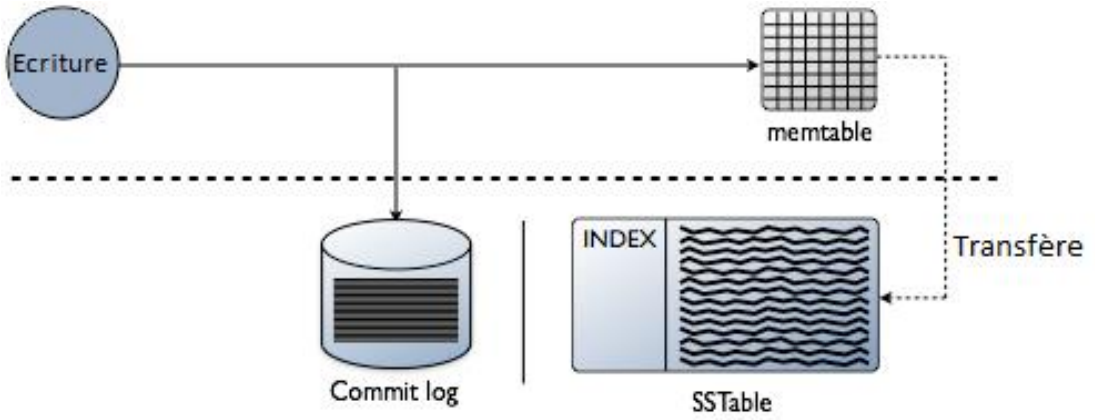

Figure 13 Cheminement de l'écriture d'une donnée [26]

II.11 – SSTables

Les données sont stockées au sein des SSTables [9, p425], elles sont immuables, au lieu de récrire une donnée, Cassandra enregistre une nouvelle version de la donnée. Les données devant être supprimées sont marquées comme tombstone.

Avec le temps le nombre de données augmente, pour garantir la bonne santé de la base de données, les tables SSTables sont fusionnées et les anciennes données supprimées, ce mécanisme s'appelle le compactage.

II.12 - Lecture et écriture fonctionnement interne

Lorsque Cassandra reçoit une requête la première étape est de déterminer le nœud coordinateur, soit Cassandra utilise sa topologie pour le déterminer, soit le client a choisi de spécifier un nœud en particulier.

II.12.1 - Client et nœud

Un client exécutant des requêtes en écriture ou en lecture peut être configuré pour envoyer ses requêtes à un nœud spécifique du cluster. Le choix du nœud s'effectue au niveau de la configuration du client, le client peut configurer un ou plusieurs nœuds comme choit de coordinateur ou bien laissé Cassandre gérer le choit du nœud. Ce nœud prendra le rôle de coordinateur pour la requête reçu.

Un nœud coordinateur agira comme un proxy entre le client et le(s) nœud(s) possédant les données. En se basant sur la configuration du cluster, il déterminera quel nœud au sein de l'anneau recevra la requête. En fonction du niveau de cohérence souhaité, le client pourra se contenter que d'une seule réponse d'un nœud.

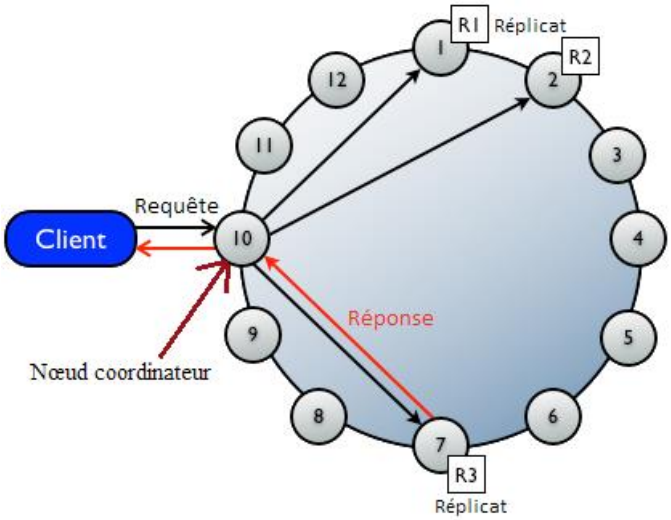

Figure 14 Chemin d'une requête en lecture

II.12.2 - Requête en écriture

Le coordinateur envoie la requête en écriture sur tous les nœuds qui possède la ligne concernée, chaque nœud va traiter la requête en suivant ces étapes :

- Écriture de la donnée dans le commitlog puis la memtable
- Envoyer un acquittement pour valider le succès de l'opération

En fonction du niveau de cohérence choisit lors de l'émission de la requête un pourcentage plus ou moins élevé de nœuds devront acquitter le succès de l'opération.

Si le niveau de cohérence accepte un seul acquittement pour valider l'opération mais qu'un autre nœud n'a pas réussi à effectuer l'opération d'écriture la cohérence du nœud serait atteinte ultérieurement avec le mécanisme interne de réparation.

II.13 – Fonctionnalité

II.13.1 – Cache

Le cache est un système permettant de satisfaire les requêtes en lecture plus rapidement en contrepartie d'une utilisation plus élevée de la mémoire. Il peut être activé globalement via le fichier de configuration de Cassandra ou table par table en utilisant CQL.

Il en existe quatre types [13, p499], le cache :

- Par clé
- Par ligne
- Par chunk
- Par comptage

Le cache par clé met en mémoire une liste des clés de partitions en relation avec les lignes. Ce type de cache ne consomme que peu de mémoire et augmente grandement les performances, il est activé par défaut.

Le cache par ligne permet de mettre en cache un nombre configurable de lignes par partitions. Il est consommateur de ressources et peu provoqué des baisses de performance si mal utilisé. Il est particulièrement efficient lorsqu'il est activé pour lire un nombre faible de données très rapidement, cependant les performances sur des gros volumes seront dégradées.

La mise en cache par chunk, place un groupe de données qui a été lu dans une SSTables qui était compressé, il est activé par défaut.

Le comptage est le seul mécanisme de cache qui ne peut pas être configuré par table mais uniquement globalement, il est utilisé lorsqu'une colonne est définie en mode comptage, cette colonne s'incrémente automatiquement, elle peut par exemple correspondre à un nombre de visiteur d'un site web. Cassandra supportant mal les modifications fréquentes d'une donnée, le cache de type comptage a été créer pour pallier ce problème.

II.13.2 - Compactage

Le compactage permet d'augmenter les performances en réduisant le nombre de fichiers à lire pour récupérer une donnée.

Ce mécanisme est automatique, un compactage manuel est possible mais à partir du moment où on exécute manuellement un compactage il faudra le faire de manière manuelle à chaque fois.

Ce mécanisme va récupérer dans les SSTables toutes les versions disponibles des lignes pour former une nouvelle ligne en utilisant seulement la dernière version disponible. Toutes les données marquées comme tombstone sont supprimées et il regroupe les colonnes dispersées qui devraient être stockées ensemble dans de nouvelle SSTables.

Lors du compactage l'espace disque utilisé augmente du faîte de la création de SSTables temporaires, à la fin du processus on bénéficiera d'un gain d'espace disque induite par la suppression des anciennes versions des données et de celles marquées comme tombstone.

Cassandra commence déjà à utiliser les dernières données disponibles dans les nouvelles SSTables même si le compactage est toujours en cours d'exécution

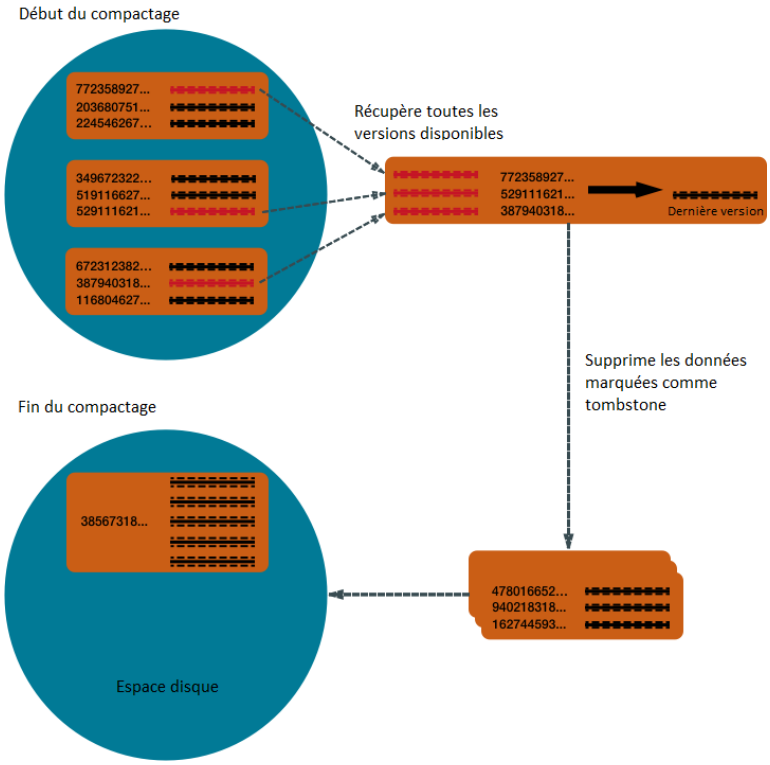

Figure 15 Processus de compactage des données [27]

II.14 - Maintenance & Opérationnel

II.14.1 – Nœud Seed

Le seed est utilisé par les nœuds pour la découverte d'autres nœuds, on peut configurer une liste de nœud seed, ces nœuds auront plus de chances d'être contactés via le protocole gossip pour établir la topologie du réseau. Grâce à cela les nœuds peuvent détecter plus rapidement les changements d'états des autres nœuds. Il est recommandé d'avoir au moins deux nœuds par datacenter listés comme seed et de synchroniser les nœuds avec la liste [14], il est déconseillé de mettre tous les nœuds dans la liste de seed.

II.14.2 - Mécanisme de réparation

Avec le temps, certains nœuds ont pu manquer des modifications et créer une incohérence des données. Chaque cluster Cassandra possède des mécanismes de réparation des nœuds pour rétablir la cohérence entre eux, trois types de réparation sont possibles :

- Le « Hinted Handoff », il agit lorsque lors de l'écriture d'une donnée un nœud ne répond pas, le coordinateur garde alors la donnée et la retransmet au nœud dès qu'il redevient disponible.

- La réparation lors de la lecture améliore la cohérence des données dans un cluster à chaque lecture. Lorsqu'une requête demande à lire des données, le coordinateur récupère ces données d'un certain nombre de nœuds défini par le facteur de réplication, et les compare entre elles, si un nœud ne possède pas la dernière version de la donnée, le coordinateur transmettra la dernière version au nœud concerné.
- L'« Anti-entropy repair », est appelé avec la commande « nodetool repair » , elle permet de revérifier et de corriger la cohérence des données à l'intérieur d'un cluster. Cette commande va comparer toutes les données et leurs copies et corriger les incohérences entre les nœuds.

Les mécanismes de « Hinted Handoff » et de réparation lors de la lecture, qui gère automatiquement la cohérence des données ne sont pas infaillible et ne dispense pas d'effectuer des réparations manuelles avec l'« Anti-entropy repair »,.

La gestion de la réparation peut s'effectuer de manière graphique via une interface web en utilisant le logiciel open source Reaper.

II.14.3 – Sauvegarde

« Tout ce qui est susceptible d'aller mal ira mal. », La loi de Murphy.

Cassandra étant distribuée, décentralisée et ses données répliquées, lorsque l'on possède un grand nombre de nœud distribué dans différents datacenters, on peut se poser la question de la nécessité d'avoir une sauvegarde de la base de données.

Personne n'est à l'abri d'une erreur humaine, de configuration de cluster ou bien d'une panne matérielle massive, les événements les plus fréquents qui mènent aux besoins de sauvegarde sont :

- Erreur faite lors d'une requête en écriture
- Suppression de données accidentelles
- Une panne matérielle
- Une corruption des données
- Le transfert d'un cluster vers un nouveau serveur

Cassandra possède deux types de sauvegarde [13, p 462][15] :

- Les snapshots
- Les sauvegardes incrémentales

Un snapshot est une copie de la définition et des données d'une table SSTable à un moment donné. Un snapshot peut être manuel ou déclenché automatiquement avant un compactage ou la suppression d'une table. Si l'option « snapshot_before_compaction » est activée un snapshot sera créé avant chaque compactage. Les snapshots sont stockés dans le sous-dossier /Keyspace/Table/snapshots.

Cassandra possède un système de snapshots automatique activé par défaut, l'option « auto_snapshot » qui crée automatiquement un snapshot en cas de suppression d'une table. À cause decette option la suppression effective d'une table peut être mise en attente pendant 60 secondes par défaut le temps de faire le snapshot.

Les sauvegardes incrémentales sont une copie d'une table SSTable qui survient au moment où la memtable est transférée sur le disque dans la SSTable, les tables système qui contiennent les informations sur un Keyspace sont elles aussi sauvegardées. Les sauvegardes incrémentales ne sont pas activées par défaut. Elles sont automatiques et sont présentes dans le sous-dossier /Keyspace/Table/backups du serveur.

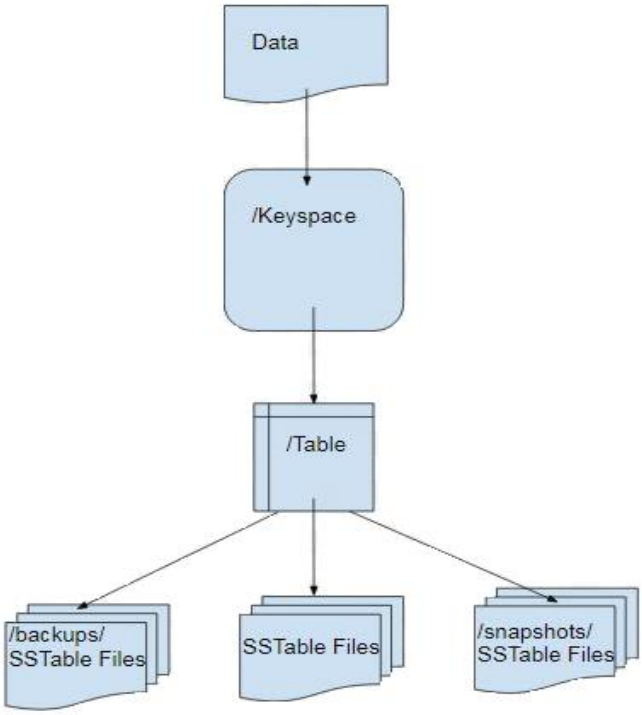

Figure 16 Structure fichier Backups et Snapshots [28]

Les points importants à prendre en compte dans sa stratégie de gestion des sauvegardes et de restauration sont :

- L'impact du stockage des sauvegardes
- La perte de données maximale admissible
- La durée maximale d'interruption admissible
- Les performances de l'application pendant la sauvegarde

II.14.4 - Monitoring

Cassandra utilise une librairie Java nommé « Dropwizard metrics » [16] pour fournir des métriques sur ses nœuds. Ces métriques peuvent être récupérées via JMX ou bien envoyées vers un autre système de monitoring via la configuration de la librairie. Cassandra ne fournit des métriques que nœud par nœud, pour avoir une vue d'ensemble il faudra les agréger.

II.15 - Bien utiliser Cassandra

Les points importants lors de l'utilisation de Cassandra sont :

Posséder deux ou trois nœuds seed par datacenter pour optimiser gossip, le protocole de découverte des nœuds. Mettre tous les nœuds en tant que seed réduirait les performances de gossip.

Toujours penser aux requêtes que l'on va lancer sur la base avant de créer les tables, ne pas hésiter à avoir plusieurs fois les mêmes données, mais dans des tables différentes pour servir des requêtes spécifiques.

Effectuer régulièrement des backups et snapshots et tester d'effectuer une restauration.

Pour assurer la cohérence des données, il est recommandé de lancer régulièrement le processus de réparation.

Laisse le compactage s'effectuer et ne pas le lancer manuellement car une fois lancée manuellement, les suivant devront être fait manuellement aussi.

Éviter de modifier ou de supprimer des données de manière régulière, Cassandra n'est pas adaptée pour ce type d'utilisation, il vaut mieux se tourner vers une autre base de données.

II.16 - Cas d'utilisation

Cassandra est particulièrement adaptée au projet de grande envergure, pour profiter de toutes ses fonctionnalités il faut un certain nombre de nœuds. Les très grands nombres d'écritures sont très bien gérés cependant la modification ou la suppression de données diminue fortement les performances.

Les données de type logs, série temporelle sont particulièrement adaptées. Un autre cas d'utilisation pour Cassandra est quand les requêtes ne sont pas sujettes à des changements, exécuter la même requête permet d'optimiser la structure des données à ses types de requêtes.

Cassandra n'est pas une base de données à tout faire, elle est adaptée pour résoudre des besoins spécifiques et déterminés à l'avance.

II.17 - Limitations

Les changements fréquents des données et les suppressions régulières sont très couteux en matière de performance. Il n'y a pas de différence entre une insertion de données et une modification, toutes les versions de ses changements sont gardées jusqu'au prochain compactage, lors d'une lecture Cassandra va rechercher la dernière donnée à jour et si plusieurs versions remontent de nœuds différents, une réconciliation sera nécessaire pour déterminer la dernière version, cela provoquera de la latence sur la requête en lecture.

Cassandra et la gestion de file de message ne sont pas compatibles, un grand nombre d'écriture et suppression successive générera beaucoup de donnée non à jour et de tombstone pouvant amener Cassandra à renvoyer des erreurs. Pour ce type de besoin il vaut mieux se

tourner vers des solutions comme RabbitMQ, Apache Kafka ou une solution Cloud comme Amazon SQS.

Toute application n'utilisant pas des requêtes prédéterminées à l'avance aura des performances médiocres avec Cassandra, les familles de colonnes doivent être pensées et créer pour les requêtes.

Les solutions faisant régulière des scans complets d'une table avec des requêtes du type « select * » sans clause where peuvent causer une panne sur le nœud coordinateur, cela est dû aux faites que le coordinateur va devoir vérifier sur tous les autres nœuds la présence des données et agréger les résultats, sur qui peut représenter un énorme volume de données que le nœud coordinateur ne pourra pas supporter.

Cassandra étant particulièrement adaptée pour des gros volumes de données répliquées sur plusieurs serveurs, il n'est pas judicieux d'utiliser Cassandra pour des petits projets, en effet, la maintenance et l'optimisation de Cassandra nécessite beaucoup de savoir et de temps. Il faut aux minimums trois serveurs pour tirer parti de ses fonctionnalités et les besoins matériels de ces serveurs sont élevés, un serveur de production nécessite 8 cœurs CPU ou plus, ainsi que 32GB de RAM minimum [30]. Une piste à explorer pour l'utilisation de Cassandra sans avoir à gérer la partie serveur logiciel et matériel, est les nouveaux services serverless sur le cloud comme Keyspaces d'AWS par exemple [17]

II.18 - Contributeur au projet

Cassandra fait partie des projets détenus par l'Apache Software Foundation, en tant que tel elle est open source. Une entreprise en particulier est très active dans la contribution au projet, cette entreprise privée nommée DataStax à rejoint le projet très tôt dans son développement et y a beaucoup contribué. Elle met en avant le fait qu'elle est responsable de 85% des commits, Datastax a construit son modèle économique en fournissant principalement des services autour de Cassandra tout en développant sa version propriétaire commerciale DataStax Enterprise (DSE).

Cette très forte présence d'une entreprise sur un projet open source a commencé à émettre des questionnements sur la gouvernance du projet [18], au moment de ces questionnements en 2016, un des cofondateurs de Datastax faisait partie du Comité de gestion du projet au sein de l'ASF. À la suite de ces questionnements, Datastax a pris un peu de recul et a perdu un peu de son emprise sur le projet et, le cofondateur et membre du Comité de gestion du projet à quitter sa position au sein de la fondation Apache [29].

Le projet compte aussi des membres du comité de gestion et contributeurs de différentes entreprises qui utilise la base de données comme, Apple, Netflix et Instagram [19].

Conclusion

Le monde des bases des données a su s'adapter aux nouveaux besoins et défis amenés par l'explosion du volume des données traitées, des temps de latences devant être de plus en plus court et, une forte demande pour de la haute disponibilité et la réplication des systèmes de base de données. Le modèle NoSQL depuis sa renaissance en 2009 [2][5] c'est imposé face à ces nouveaux besoins, pour chaque problème spécifique, une base de données utilisant le schéma de données le plus adapté est née.

Comme le démontre le théorème CAP [4], un système ne peut réunir que deux des propriétés suivantes : cohérence, disponibilité et tolérances aux pannes. NoSQL à fait le choix de concentrer sur la disponibilité et la tolérance aux pannes, domaine dans lequel des systèmes de base de données comme Cassandra excelle, cependant NoSQL n'a pas vocation à remplacer le modèle SQL, chacun répondant à des besoins différents, il est important de connaître ses attentes pour ensuite faire son choix de manière avisée.

Apache Cassandra de par sa conception enrichie grandement le panel des cas d'utilisation du modèle NoSQL, depuis sa création au sein de Facebook puis de sa reprise par la fondation Apache en 2009 [5], elle a su s'imposer comme l'une des bases de données NoSQL les plus utilisées. Grâce à son aspect décentralisé et répliqué, elle fournit un haut taux de disponibilité et une très grande tolérance aux pannes. Avec son système de nœud et de partitionnement elle fournit de bonne performance en lecture et est excellente en vitesse d'écriture.

Cassandra est faite pour certains cas d'utilisation précis, elle n'a pas pour objectif d'être une base de données à tout faire. Elle supporte très mal la suppression et la modification de données de manière répété, son coût de maintenance est élevé et nécessite de bonne connaissance sur son fonctionnement interne. Elle est aussi inadaptée au projet de petite envergure.

Ce qui fait la force de Cassandra est aussi un grand frein à son adoption, pour supporter son modèle répliqué et décentralisé beaucoup de mécanismes internes ont été mis en place, complexifiant ainsi sa gestion et sa maintenance. Étant extensible linéairement elle est particulièrement adéquate avec une utilisation sur le cloud, des services hébergés offrant des solutions clés en main pour Cassandra commencent à voir le jour [17][20].

Est-ce que ces services hébergés sur le cloud seront fournir le même niveau de performance qu'un cluster auto-hébergé et ainsi permettre aux utilisateurs de se libérer de la contrainte de la maintenance complexe et coûteuse de Cassandra.

Bibliographie

Dernière consultation des sites : Mai 2021

[1] Entreprise utilisant Cassandra

https://cassandra.apache.org/case-studies/

[2] Réutilisation du nom NoSQL pour Not Only SQL en 2009

https://web.archive.org/web/20110710205509/http://nosql.eventbrite.com/

[3] Cassandra Query Language :

https://docs.datastax.com/en/archived/cql/3.3/cql/cqlIntro.html

[4] Théorème CAP

https://en.wikipedia.org/wiki/CAP_theorem

[5] Date de reprise de Cassandra par Apache :

https://incubator.apache.org/projects/

[6] Version stable Cassandra, Mai 2021.

https://cassandra.apache.org/download/

[7] Cassandre utilise gossip

https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/architecture/archGossipAbout.html

https://cassandra.apache.org/doc/latest/faq/#why-cant-list-all

[8] Architecture Cassandra

https://docs.datastax.com/en/cql-oss/3.3/cql/ddl/dataModelingApproach.html

[9] A.PLOETZ - D.KANDHARE - S.KADAMBI – Xun (Brian).WU - Seven NoSQL Databases in a Week - Packt Publishing - 2018.

[10] Architecture Cassandra

https://cassandra.apache.org/doc/latest/architecture/dynamo.html

[11] Snitch

https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/architecture/archSnitchesAbout.html

[12] Écriture donnée Cassandra

https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlHowDataWritten.html

[13] J.CARPENTER - E.HEWITT - Cassandra: The Definitive Guide, Third Edition - O'Reilly Media - 2020.

[14] Le nombre de seed recommandés pour Cassandra

https://cassandra.apache.org/doc/latest/faq/#what-are-seeds

[15] Sauvegarde, snapshot et backups

https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/operations/opsBackupRestore.html

[16] Monitoring Dropwizard Metrics utilisé par Cassandra

https://cassandra.apache.org/doc/latest/operating/metrics.html

[17] Service Serverless Cassandra AWS Keyspaces

https://docs.aws.amazon.com/keyspaces/latest/devguide/what-is-keyspaces.html

[18] Gouvernance Cassandra, Chris Mattmann, Membre du Conseil d'administration de l'Apache Software Foundation en 2016

https://mail-archives.apache.org/mod_mbox/cassandra-dev/201606.mbox/browser

[19] Liste membres du Comité de gestion et contributeurs Cassandra

https://projects.apache.org/committee.html?cassandra

[20] Offre cloud Cassandra Datastax

https://www.datastax.com/products/datastax-astra

[22] Figure popularité base de données NoSQL

https://db-engines.com/en/ranking_trend

[23] Figure, Clé-valeur NoSQL

https://en.wikipedia.org/wiki/Key%E2%80%93value_database#/media/File:KeyValue.PNG

[24] Figure Keyspace Cassandra

https://ftp.utcluj.ro/pub/users/civan/IBD/3_EVALUARE/1_Referat/Resurse/B_2017_Mongo_cassandra.pdf

[25] Figure Exemple triage avec clé partition et clustering colonne

https://blog.ippon.tech/modeling-data-with-cassandra-what-cql-hides-away-from-you/

[26] Figure écriture des données Cassandre

https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlHowDataWritten.html

[27] Figure Compactage

https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlHowDataMaintain.html#dmlHowDataMaintain__dml-compaction

[28] Figure Backups et Snapshots Cassandra

https://cassandra.apache.org/doc/latest/operating/backups.html

[29] Co-Fondateur de Datastax se retire du comité d'Apache

https://sdtimes.com/apache/jonathan-ellis-steps-cassandra-project/

[30] Besoin matériel Cassandra

https://cassandra.apache.org/doc/latest/operating/hardware.html

Table des matières