Le NoSQL - Cassandra

Thèse Professionnelle

Xavier MALETRAS

27/05/2012

Ce document présente la technologie NoSQL au travers de l'utilisation du projet Cassandra. Il présente des situations ainsi que des préconisations d'utilisations.

Sommaire

I. Introduction

Depuis toujours, l'homme n'a cessé de se développer et de développer son savoir et ses connaissances sur l'environnement qui l'entoure. Le développement est passé par le vécu et l'expérience. Afin de profiter de cette expérience, il a mis en place des outils dans le but de stocker et traiter les informations liées à cette expérience.

De nos jours, la donnée de façon générale a une importance capitale au bon fonctionnement d'une société. Les outils et procédés de mesure, de gestion et d'analyse de données forment la base à partir de laquelle se développe la compréhension de notre environnement. L'informatique a pris une large place dans ces procédés et est devenue, aujourd'hui, indispensable pour tout ce qui concerne la gestion de l'information en entreprise.

Les données ne sont pas toujours identifiées comme un enjeu stratégique par les entreprises et les organisations gouvernementales. Pourtant, à un moment où l'on parle de plus en plus d'économie du savoir, on constate que les informations qui constituent le cœur de métier des entreprises deviennent stratégiques. La question de la gestion de données est donc devenue incontournable pour toutes ces entreprises.
Des outils et méthodes spécifiques ont été créés dans le but de pouvoir stocker, gérer et éventuellement archiver toutes ces informations. Tout au long de l'évolution du domaine informatique, ces outils se sont développés et sont utilisés par les environnements disposant d'un système informatisé.

Les données sont donc récoltées, gérées, analysées, suivant un processus souvent standardisé à l'aide d'un système de gestion de bases de données, non relationnelle dans un premier temps puis relationnelle ensuite. Il existe plusieurs éditeurs qui se partagent le marché.

Depuis quelques années est apparu un nouveau mouvement issu d'une réflexion sur les systèmes existants ainsi que leurs limites et contraintes, le NoSQL (« Not Only SQL » ou « Pas seulement SQL » en français). Le fruit de ce mouvement et de ces réflexions a abouti aux développements de plusieurs projets aujourd'hui utilisés par de gros sites sur internet.

II. Système de gestion de bases de données

Les avancées de l'électronique et de l'informatique ont enrichi la pratique de la récolte et de la gestion des données. La constante est l'accroissement des capacités de traitement, tant au niveau de l'acquisition que du stockage ou de l'accès aux données.

Dans les entreprises, on manipule souvent des données ayant la même structure. Par exemple, pour une personne, nous allons enregistrer son nom, son prénom, son sexe, sa date de naissance, son adresse, sa fonction. Ses informations constituent une base de données si elles sont gérées par des moyens informatiques.

Une base de données : c'est un ensemble d'informations connexes enregistrées dans un dispositif informatique. Dans une base de données relationnelle, les informations sont stockées sous forme de groupe de valeurs : les enregistrements. Un ensemble d'enregistrements relatif à un sujet forme une relation et est stocké dans une table. La base de données comporte une ou plusieurs tables et les sujets sont connexes.

Toutes informations utiles à une entreprise sont stockées dans des bases de données. Il existe plusieurs types de bases de données, bases de données, banques de données, bases de données relationnelle, un entrepôt de données (Data Warehouse), etc. Dans ce document, nous nous intéresserons essentiellement aux bases de données dites classiques et aux bases de données relationnelles.

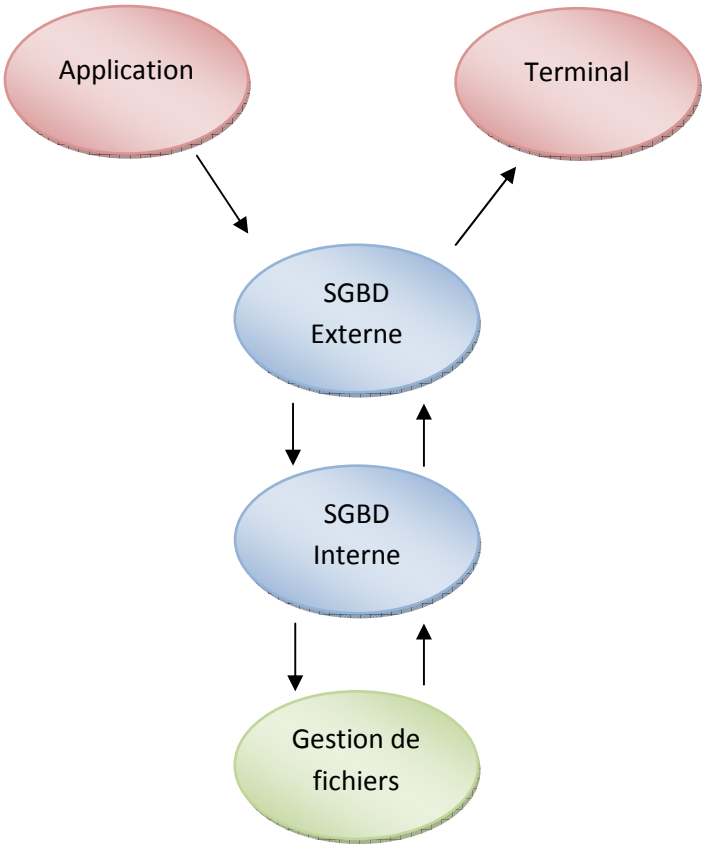
Une base de données relationnelle : c'est un stock d'informations décomposées et organisées dans des matrices appelées relations ou tables conformément au modèle de données relationnel. Le contenu de la base de données peut ainsi être synthétisé par des opérations d'algèbre relationnelle telles que l'intersection, la jointure ou le produit cartésien.

Afin de pouvoir contrôler les données ainsi que les utilisateurs, le besoin d'un système de gestion s'est vite fait ressentir. C'est en partant de ce constat qu'on été créés des systèmes de gestion de bases de données.

Système de gestion de base de données (SGBD ou DBMS en anglais) : c'est un ensemble de logiciels informatiques qui sert à la manipulation des bases de données. Il sert à effectuer des opérations ordinaires telles que consulter, modifier, construire, organiser, transformer, copier, sauvegarder ou restaurer des bases de données. Il est souvent utilisé par d'autres logiciels ainsi que les administrateurs ou les développeurs.

A. Principe de fonctionnement

Un système de gestion de base de données permet la gestion de bases de données multiples :



Le SGBD peut se décomposer en trois sous-systèmes :

- Le système de gestion de fichiers :
  Il permet le stockage des informations sur un support physique

- Le SGBD interne :
  Il gère l'ordonnancement des informations

- Le SGBD externe :
  Il représente l'interface avec l'utilisateur

B. Le théorème de CAP

CAP est l'acronyme de « Consistency, Availability and Partition Tolerance », ou « Cohérence, Disponibilité et Résistance au morcellement ». Ce théorème est aussi connu sous le nom de « théorème de Brewer ».

Ce théorème explique qu'il est impossible qu'un système d'information à calcul distribué satisfasse à la fois aux trois contraintes suivantes :

- **Cohérence** : tous les nœuds du système voient exactement les mêmes données à chaque moment.
- **Disponibilité** : les données sont disponibles autant que possible, si possible même en cas de crash d'un serveur.
- **Résistance au morcellement** : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement, ce qui veut dire qu'en cas de morcellement en sous-réseaux, chaque nœud doit pouvoir fonctionner de manière autonome.

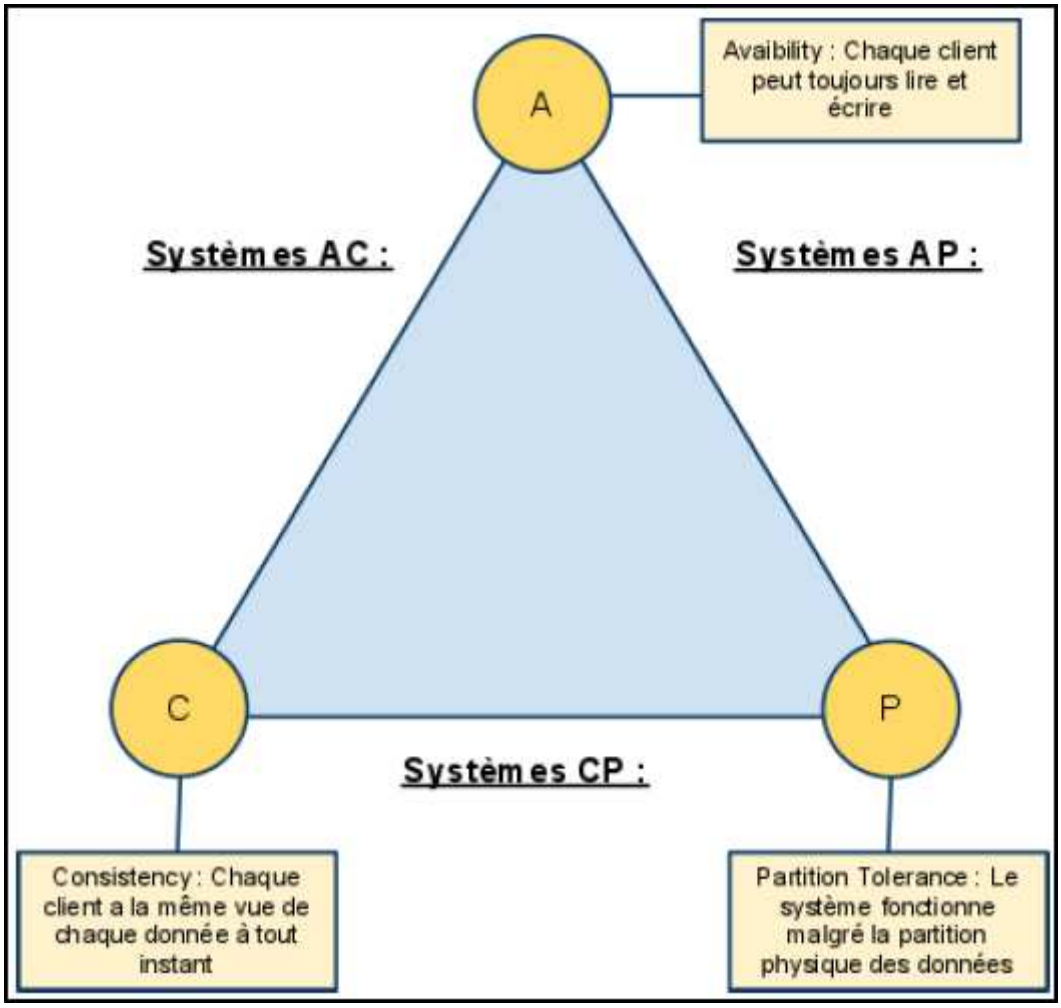
En règle générale, lors de l'utilisation d'un SGBDR, seuls deux des trois contraintes citées précédemment sont respectées.

En pratique, la cohérence ne peut pas être véritablement réalisée, notamment dans le cas d'accès concurrentiel aux données. Dans ce cas, les données sont modifiées par un client, et les autres clients connectés devront attendre la fin des modifications pour avoir accès aux données modifiées. Pendant le temps que durent les modifications, l'ensemble des clients ne voit donc pas les mêmes données, et le principe de cohérence n'est pas respecté. Il est toutefois respecté à terme, puisqu'une fois les transactions terminées et les ajouts/modifications enregistrés, l'ensemble des clients voient à nouveau les mêmes données.

Le principe de disponibilité correspond au besoin de rendre les données accessibles aux utilisateurs en toute circonstance. Le respect de ce principe est rendu possible par l'utilisation de systèmes de sauvegarde et de réplication. Ces systèmes peuvent être inclus dans les solutions de SGBD.

Enfin la résistance au morcellement est le fait que, dans le cas où une partie su système d'information venait à être défaillante au point d'être isolé du reste du système, le reste du système doit pouvoir continuer à être accessible et à fonctionner de façon normale.

Le théorème de CAP explique que, puisqu'il est impossible de satisfaire à ces trois contraintes dans le même temps, il faut en choisir deux qui serviront de base au système de gestion des données. On obtient ainsi trois types de systèmes : AP, CP et AC, dont l'orientation est guidée par deux des trois principes énoncés par le théorème de CAP.



Les 3 types de systèmes promus par le théorème de CAP.

Les bases de données relationnelles gèrent principalement la cohérence et la disponibilité, ce qui forme des systèmes AC. Les bases NoSQL sont utilisés pour les systèmes de types AP ou CP.

C. Le principe d'acidité

Le fait que la plupart des SBGD fonctionnent en mode « transactionnel » leur impose le respect des contraintes dites « ACID ». ACID est un acronyme pour « Atomicity, Consistency, Isolation and Durability ». Le respect de ces contraintes permet de s'assurer que les transactions réalisées se font de façon correcte et en toute sécurité.

- **Atomicity :** Cela signifie que les mises à jour de la base de données doivent être atomiques, c'est-à-dire qu'elles doivent être totalement réalisées ou pas du tout. Par exemple, sur 5000 lignes devant être modifiées au sein d'une même transaction, si la modification d'une seule échoue, alors la transaction entière doit être annulée. C'est primordial, car chaque ligne modifiée peut dépendre du contexte de modification d'une autre, et toute rupture de ce contexte pourrait engendrer une incohérence des données de la base.

- **Consistency :** Cela signifie que les modifications apportées à la base doivent être valides, en accord avec l'ensemble de la base et de ses contraintes d'intégrité. Si un changement enfreint l'intégrité des données, alors soit le système doit modifier les données dépendantes, comme dans le cas classique d'une suppression en cascade, soit la transaction doit être interdite.

- **Isolation :** Cela signifie que les transactions lancées au même moment ne doivent jamais interférer entre elles, ni même agir selon le fonctionnement de chacune. Par exemple, si une requête est lancée alors qu'une transaction est en cours, le résultat de celle-ci ne peut montrer que l'état original ou final d'une donnée, mais pas l'état intermédiaire. De fait, les transactions doivent s'enchaîner les unes à la suite des autres, et non de manière concurrentielle.

- **Durability :** Cela signifie que toutes les transactions sont lancées de manière définitive. Une base ne doit pas afficher le succès d'une transaction, pour ensuite remettre les données modifiées dans leur état initial. Pour ce faire, toute transaction est sauvegardée dans un fichier journal, de sorte que si un problème survient empêchant sa validation complète, la transaction pourra être correctement terminée lors de la disponibilité du système.

Dans un contexte centralisé, les contraintes ACID sont relativement faciles à satisfaire. Par contre, dans un environnement distribué, il faut aussi distribuer les traitements de données entre les serveurs, ce qui rend difficile de satisfaire les contraintes ACID sans pertes de performance. Les systèmes de données NoSQL sont construits sans tenir compte des recommandations ACID, quitte à ne pas proposer de fonctionnalités transactionnelles.

III. Les limites des SGBDR :

Le modèle relationnel est fondé sur un modèle mathématique solide s'appuyant sur la logique des prédicats du premier ordre. Il s'appuie sur des concepts simples qui font sa force en même temps que sa faiblesse. Nous expliquerons ici les principales limites des SGBDR.

- **Modélisation des entités réelles :** les relations ne correspondent pas toujours à des entités du monde réel (notamment pour les objets complexes). Les processus de normalisation et de décomposition des schémas relationnels entraînent la multiplication des relations, et par conséquent des opérations de jointure lors du traitement des requêtes.

- **Surcharge sémantique :** le modèle relationnel s'appuie sur un seul concept (la relation) pour modéliser à la fois les entités et les associations / relations entre ces entités. Il existe donc un décalage entre la réalité et sa représentation abstraite.

- **Contraintes d'intégrité :** les SGBD relationnels sont limités à des contrôles de cohérence simples. Il est donc difficile de modéliser des contraintes réelles correspondant aux données d'une entreprise. Ce problème n'est que partiellement résolu avec SQL-2 qui permet d'exprimer des contraintes dans la partie langage de définition.

- **Types de données :** ils sont limités à des types simples (entiers, réels, chaînes de caractères), les seuls types étendus se limitant à l'expression de dates ou de données financières, ainsi que des conteneurs binaires de grande dimension (*BLOB*, pour Binary Large OBjects) qui permettent de stocker des images ainsi que des fichiers audios ou vidéos. Ces BLOBs ne sont toutefois pas suffisants pour représenter des données complexes (pas de structure), les mécanismes de contrôle BD sont inexistants, et le langage de requêtes (SQL) ne possède pas les opérateurs correspondant aux objets stockés dans ces BLOBs.

- **Langage de manipulation :** il est limité aux opérateurs de base du langage SQL, qu'il n'est pas possible d'étendre à de nouveaux opérateurs, même avec SQL-2.

- **Requêtes récursives :** il est possible, avec des requêtes SQL imbriquées, de travailler sur des relations de type parent-enfant, mais il est impossible de travailler sur des relations de type ancêtre (nombre de niveaux inconnu). Pour traiter ce type de problème, les requêtes doivent être insérées dans un langage de programmation ("langage hôte").

- **Incompatibilités de types (“*impedance mismatch*”) :** elles apparaissent lors de l’utilisation de SQL dans un langage hôte (pour la représentation des nombres, par exemple). Des conversions de types sont nécessaires, et peuvent s’avérer coûteuses en temps et parfois difficiles à mettre en œuvre.

- **Scalabilité limitée :** atteinte par les groupes comme Yahoo!, Google, Facebook, Twitter ou LinkedIn, cette limite est la plus gênante pour les entreprises. Dans le cas de traitements de données en masse, le constat est simple : les SGBD relationnels ne sont pas adaptés aux environnements distribués requis par les volumes gigantesques de données et par les trafics tout aussi gigantesques générés par ces opérateurs.

Autre problème : traitement des “transactions de longue durée”, par exemple lors de la conception d’un moteur. Le problème peut devenir encore plus complexe s’il s’avère nécessaire de modifier le modèle de données en cours de projet.

Vient s’ajouter à tout cela le fait qu’un système de gestion de bases de données relationnelles est déployé sur un serveur. Celui-ci représente un point de contention (en anglais single point of failure), en cas de panne il y a une interruption de service même si dans le cadre d’un plan de reprise ce premier serveur est « backupé ».

C’est pour toutes ces raisons qu’une réflexion a été menée et que le mouvement NoSQL a été lancé.

IV. Présentation de la technologie NoSQL

A. Définition

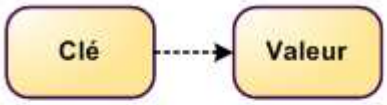
Le NoSQL, désignant « Not Only SQL », représente une catégorie de bases de données apparue au courant de l'année 2009 qui se différencie du modèle relationnel que l'on retrouve dans les systèmes de bases de données connues tels que MS SQL Server, Oracle ou PostgreSQL. Cette catégorie de produits fait le compromis d'abandonner certaines fonctionnalités classiques des SGBD relationnels au profit de la simplicité, la performance et une forte scalabilité. La scalabilité est la capacité d'un système à répondre à une demande toujours grandissante de la part des utilisateurs en termes de requêtes. Il s'agit d'une capacité de montée en charge.

B. Les différents types

Il existe aujourd'hui différents types de solution NoSQL.

1. Clé - valeur

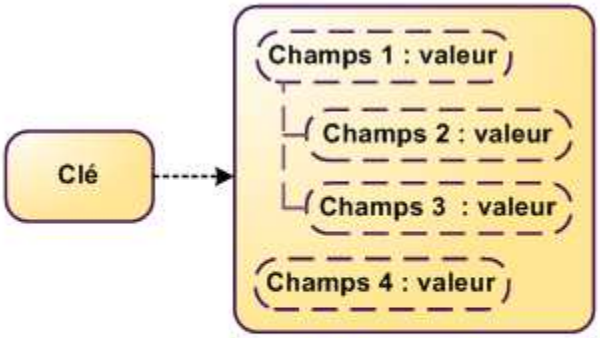
La représentation en clé-valeur est la plus simple et est très adaptée aux caches ou aux accès rapides aux informations. Elle considère la valeur stockée comme un bloc de données opaque, la base de données étant agnostique de son contenu. Ce postulat permet en général d'atteindre des performances bien supérieures dans la mesure où les lectures et écritures sont réduites à un accès disque simple.



BDD Clé-Valeur

2. Document

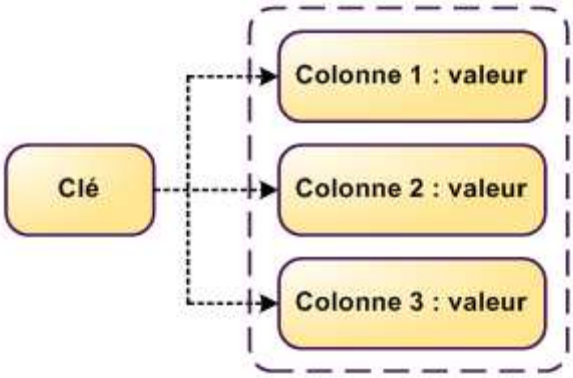
La représentation en document est particulièrement adaptée au monde du Web. Il s'agit d'une extension du concept de clé-valeur qui représente la valeur sous la forme d'un document. Un document contient des données organisées de manière hiérarchique à l'image de ce que permettent XML ou JSON. Étant consciente du contenu qu'elle stocke, la base de données peut alors effectuer des indexations de différents champs et offrir des requêtes plus élaborées.



BDD Orientée document

3. Colonne

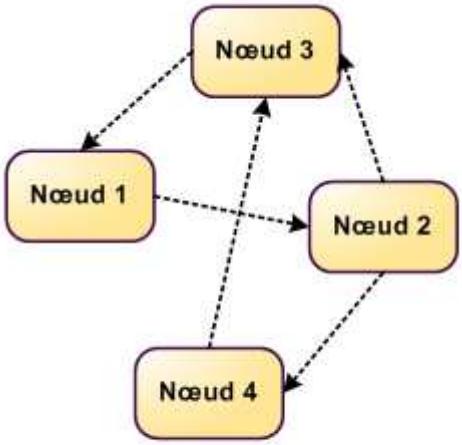
La représentation orientée colonnes s'oppose à la représentation des tables dans les bases de données relationnelles. En effet, les SGBDR manipulent les colonnes d'une ligne d'une manière statique. Les bases de données orientées colonnes ont une vision bien plus flexible permettant d'avoir des colonnes différentes pour chaque ligne et de multiplier de manière conséquente le nombre de colonnes par ligne. Il en résulte une capacité à stocker des listes d'informations pour chaque clé, et à accéder à des intervalles de colonnes.



BDD Orientée colonnes

4. Graphe

La représentation en graphe permet la modélisation, le stockage et la manipulation de données complexes liées par des relations non-triviales ou variables. Si le cas d'utilisation classique de ce type base de données est le stockage des informations des réseaux sociaux, elle est à même de modéliser de nombreuses situations du monde réel qui ne pourraient être représentées que de manière réductrice dans une base de données relationnelle.



BDD Orientée graphe

C. Le principe de BASE

Nous avons vu dans la première partie le principe d'acidité qui impose certaine contrainte au fonctionnement des systèmes de gestion de bases de données relationnelles. Le NoSQL permet d'accepter des données non consistantes. En opposition au terme d' « acid », le terme base a été choisi en référence au domaine de la chimie.

Ce principe est le résultat de la réflexion menée par le mouvement NoSQL qui s'intéressait principalement aux limites que montraient les systèmes de gestion de bases de données relationnelles.

Le principe de base se compose comme suit :

- Basically Available (Disponibilité): le système est disponible, même en cas de désynchronisation ou de panne de l'un des nœuds formant le serveur.
- Soft-state (Scalabilité): règles de cohérence faibles, ce qui permet d'être plus flexible au niveau des contrainte C'est applicable à des systèmes très vastes.
- Eventually consistent (Cohérence): la cohérence n'est pas visée après chaque opération, mais au bout d'un temps et d'un nombre d'opérations indéfini.

Les systèmes de gestion de bases de données NoSQL respectent ces trois principes qui permettent d'augmenter la disponibilité et de supporter de fortes montées en charges.

D. Les avantages et les limites

Le NoSQL présente de forts avantages face à la montée en charge par rapport aux SGBD présents sur le marché. Toutefois, cette technologie ne peut être utilisée dans tous les cas.

Le principal avantage de la technologie NoSQL réside dans le fait que cette technologie permet une forte scalabilité. En effet, depuis quelques années, la demande de montée en charge de gros sites WEB est grandissante. Les SGBD classiques ont commencé à révéler leurs limites lors de traitement de très gros volumes de données. Le NoSQL résout les problèmes de rapidité en lecture et écriture sur les données. Il y a donc un gain en termes de performances lorsqu'il n'y a pas d'accès concurrentiel aux données, comme par exemple Facebook, où il n'y a qu'un utilisateur qui peut modifier ses propres données.

Pour reprendre le théorème de CAP : les technologies NoSQL sont hautement disponibles, tolérant aux pannes réseaux mais ne répondent pas à l'aspect cohérence en temps réel des données. Toutefois, si les bases de données ne répondent pas, en direct, à la cohérence des données, celles-ci le seront a terme. Le temps de mise en cohérence dépend directement du nombre de nœuds de la base de données.

L'un des avantages du NoSQL est lié aux types de données. Le type de données est défini à l'avance et sera donc proche des besoins métiers. Il n'y aura pas ou peu de modification de format entre la base de données et les applications.

Le NoSQL n'est pas adapté à toutes les situations. Dans les environnements de production où l'intégrité des données est essentielle en temps réel et en permanence, le NoSQL est limité dans la mesure où la base de données est répartie sur différents nœuds. Si un utilisateur renseigne une donnée sur l'un des nœuds, celle-ci n'est pas dupliquée en temps réel sur l'ensemble des nœuds constituant le système.

E. Quelques acteurs majeurs

Redis est une solution open source de type clé/valeur basée sur la technologie NoSQL. Cette solution a été développée par deux développeurs indépendants, Pieter Noordhuis(californie, US) et Salvatore Sanfilippo(Sicile, Italy).

MongoDB est une solution open source de type document basée sur la technologie NoSQL. Cette solution a été développée par la société 10gen.

CouchDB est une solution open source de type document basée sur la technologie NoSQL. Cette solution a été développée en erlang par « Apache Software Foundation ».

HBase est une solution open source de type colonne basée sur la technologie NoSQL. Cette solution a été développée en java par « Apache Software Foundation ». Cette solution est utilisée par Facebook entre autre.

Cassandra est la solution la plus répandu. C'est une solution open source de type colonne basé sur la technologie NoSQL. Celle-ci a été développée en java par « Apache Software Foundation ».

BigTable est une solution propriétaire de type colonne basée sur la technologie NoSQL. Cette solution a été développée par Google. Celle-ci est utilisée par Google.

Neo4j est une solution propriétaire de type graphe basée sur la technologie NoSQL. Cette solution a été développée en Java par « Neo Technology ».

V. Cassandra

A. Présentation de Cassandra

1. Présentation

Cassandra est une base de données NoSQL orientée colonne. Elle a été conçue pour pallier au problème de performances des bases de données relationnelles, ainsi qu'a des problèmes de gestion de grandes quantités de données (rapport entre quantité et performance). Par ailleurs, elle pallie à la complexité de déploiement et au maintien de l'intégrité lors de déploiement en cluster et dans différents Datacenter. Cassandra est conçue pour gérer des quantités massives de données réparties sur plusieurs serveurs (cluster), en assurant tout particulièrement une disponibilité maximale des données et en éliminant les points individuels de défaillance. Cette solution est donc basée sur une architecture distribuée pour permettre des débits en lecture et en écriture importants. De plus, ce système possède une forte résistance à la charge de par son architecture. Cassandra reprend les concepts de 2 bases de données existantes. La première BigTable, créé par Google, pour son modèle de données orienté colonne et son mécanisme de persistance sur disque, et la seconde Dynamo, créé par Amazon, pour son architecture distribuée sans nœud maître.

2. Historique

Cassandra est un projet de base NoSQL fondé en 2008 par Facebook et est devenu Open Source en 2008. Deux ans plus tard, le projet Cassandra est repris par la fondation Apache et la même année, en 2010, le projet est abandonné par Facebook. Actuellement, Cassandra est utilisé par plusieurs sociétés, telles que Netfix ou encore Twitter, afin de traiter un grand nombre de données.

B. Fonctionnement

Architecture Distribuée

Cassandra est basée sur une architecture distribuée.

Tous les nœuds du cluster ont le même rôle et permettent donc la lecture et l'écriture des données au sein de la base NoSQL. Le client peut alors se connecter à n'importe quel nœud pour accéder aux données. Il n'y a donc plus de « Single point of failure ».

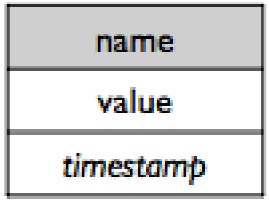
La communication entre les différents nœuds du cluster se fait en peer to peer, il s'agit du « gossip protocol ». Un utilisateur se connecte sur l'un des nœuds lors de l'ouverture de sa session sur le serveur. Ainsi si le nœud, sur lequel s'est connecté l'utilisateur, dispose de l'information que celui-ci recherche, le nœud répondra à la requête de l'utilisateur. Si toutefois ce n'est pas le cas, le nœud va communiquer avec ceux qui ont l'information pour ainsi restituer les données à l'utilisateur.

Ce système de gestion de bases de données ne gère pas le cache, il est alors nécessaire aux développeurs d'implémenter dans leurs logiciels une gestion du cache.

C. Modèle de données

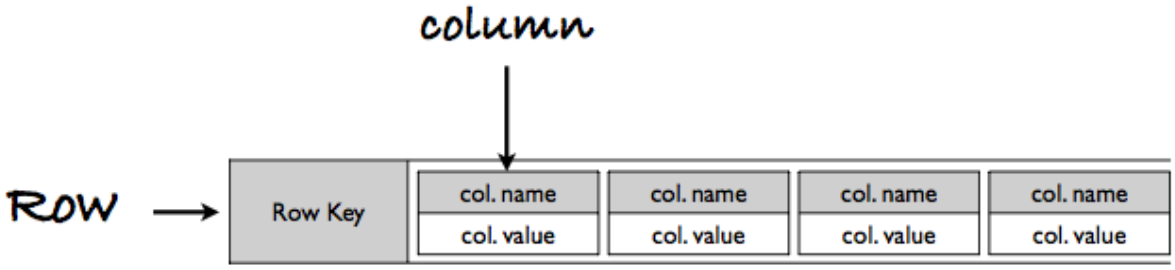
1. Les colonnes

La colonne est la plus petite unité de données. Elle est composée d'un nom (key) d'une valeur et d'un timestamp. Le nom à une taille maximum de 64Ko et la valeur a une taille maximum de 2Go. Mais la limite est que la donnée doit être contenue sur le même disque dur (sur une seule machine).

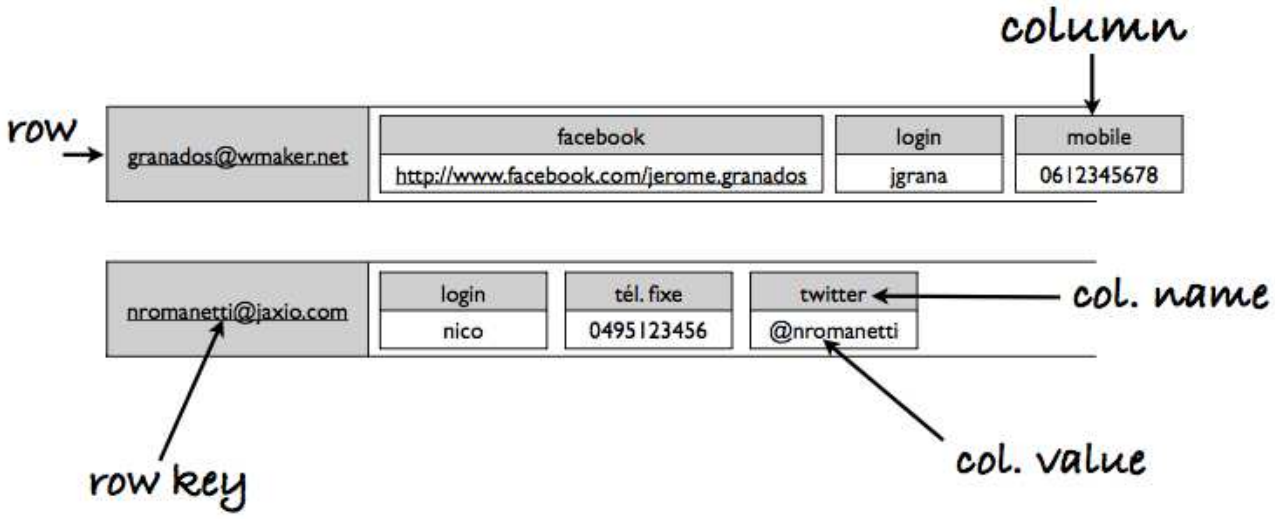


2. Les rows

Une row ou un champ est un ensemble de colonnes identifié par une clé. La clé a une longueur maximum de 64 Ko et un champ possède un maximum de 2 milliards de colonnes soit une taille maximum de 2 milliards * 2 Go soit 4 milliards de Go pour un seul champ.



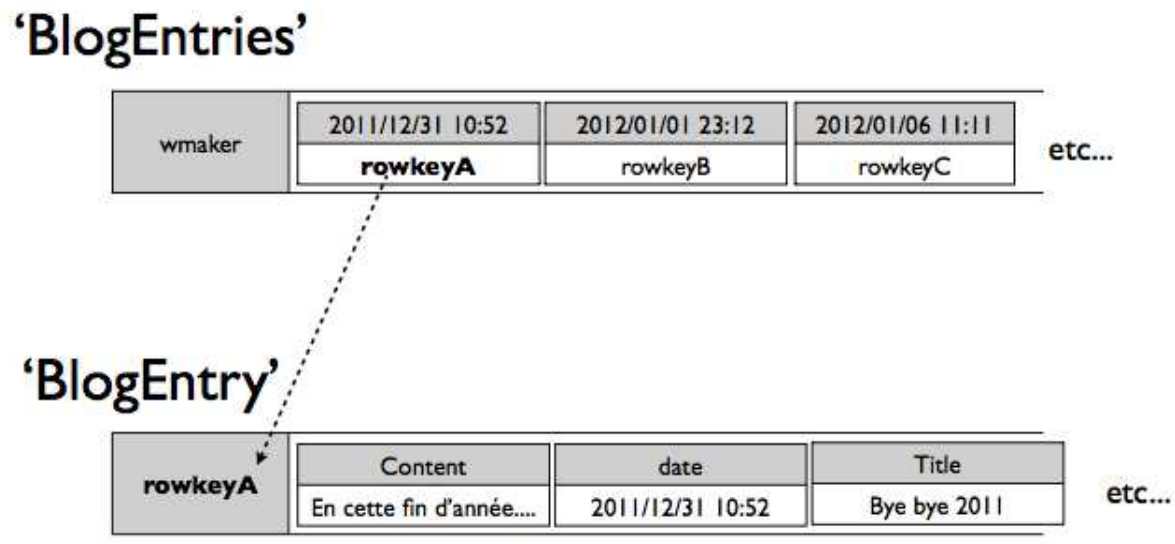
Exemple :



Il n'y a aucune contrainte sur les colonnes, elles peuvent contenir toutes sortent de données. De plus, pour optimiser les recherches et les liaisons dans l'application, il est nécessaire de déterminer une bonne structure de données et définir les bons index.

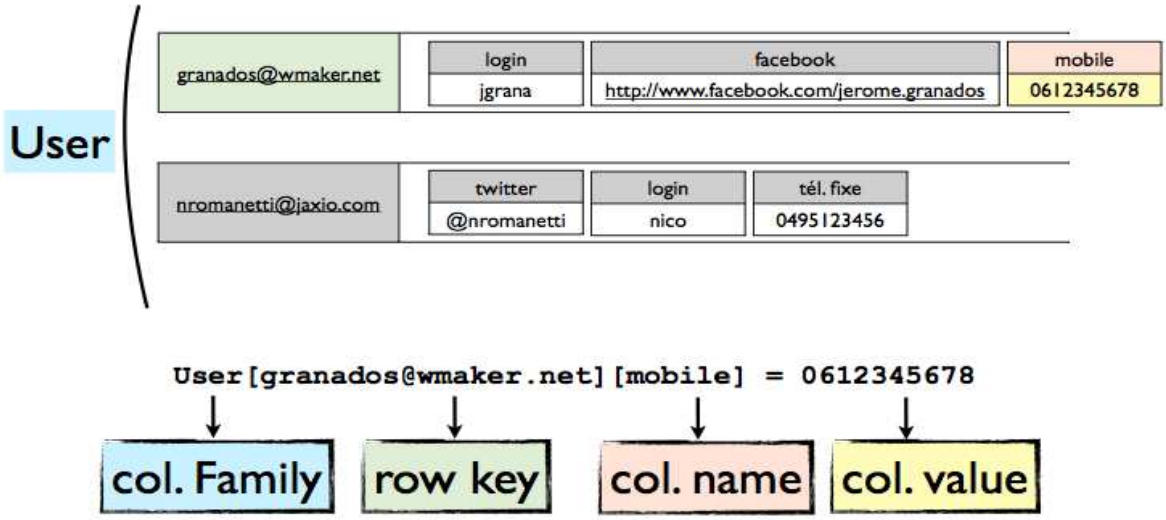
3. Les liens entre les Rows

Le plus simple et le plus efficace pour faire des liens entre les données est de créer par exemple une column family « BlogEntries» avec les différent post par utilisateur et une autre column family « BlogEntry » qui se compose de row détaillant les posts.



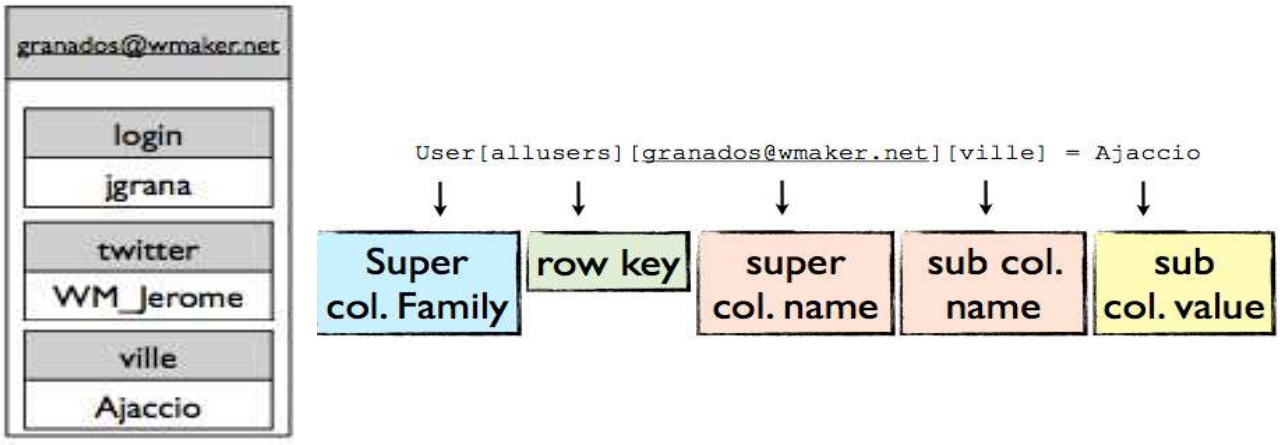
4. Les colonnes family

Une column family peut être caractérisé comme une table en BDD relationnel. Elle se compose de champs.



5. Les super colonnes

Les supers colonnes sont des colonnes contenant des colonnes, cela sert à grouper des informations dépendantes.



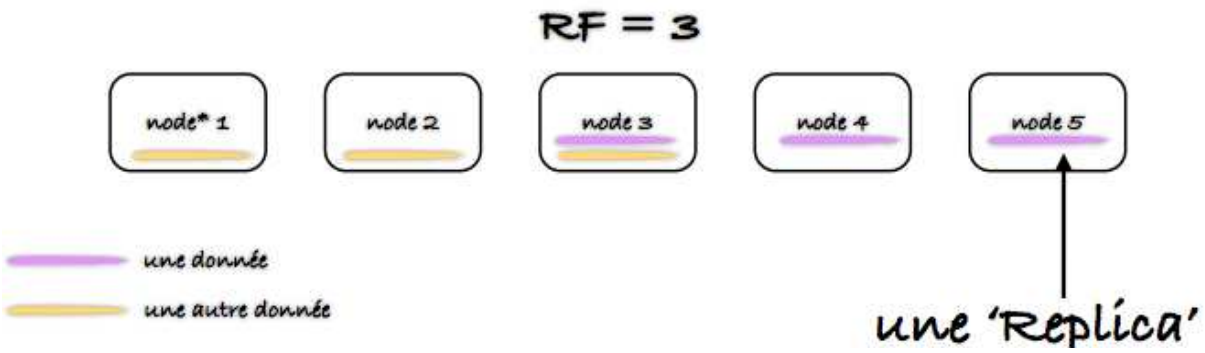
6. Les KeySpaces

Un keyspace est un ensemble de ColumnFamily. Un Keyspace définit donc uniquement (du point de vue modélisation) les ColumnFamily (et pas la structure des ColumnFamily) incluse. La notion de « ligne » n'existe pas en tant que telle : c'est une liste de Columns ou SuperColumns identifié par une clé.

D. La réplication avec Cassandra

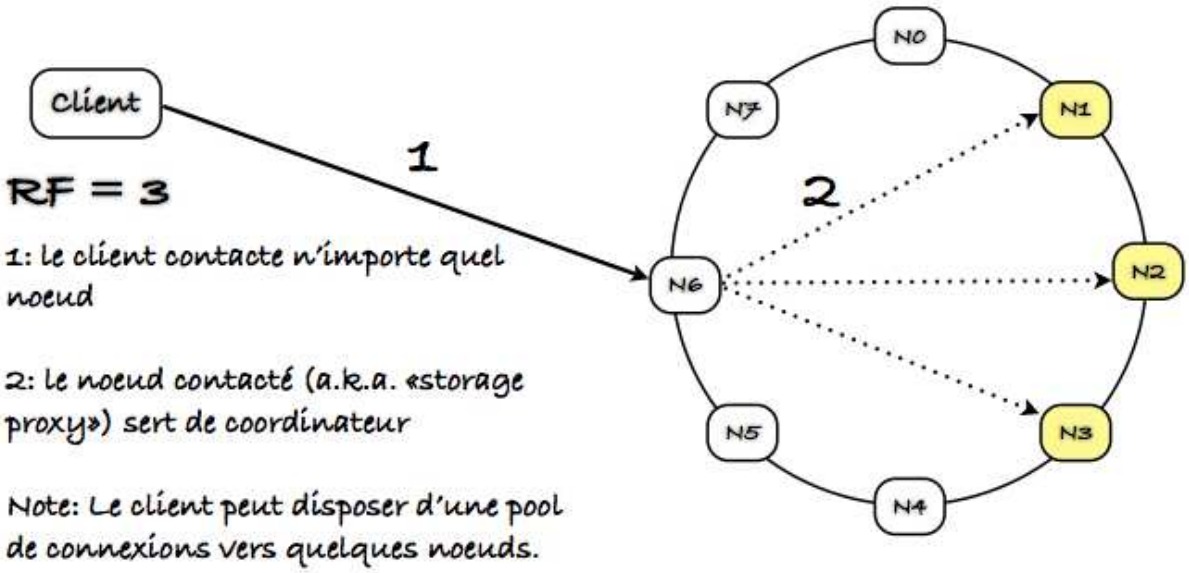
Afin d'assurer une grande disponibilité des données, il est nécessaire de répliquer les données, il s'agit du replication_factor (RF) lors de la création des keyspaces. Le RF correspond au nombre de réplicas (copies).

Exemple :



1. Ecriture

Lors de l'écriture d'une donnée sur un cluster Cassandra, selon le replication_factor, le nœud que contacte le client sert de coordinateur et écrit ensuite les données sur le nombre de nœuds indiqué (RF). Le client ne sait alors pas où est stockée la donnée.

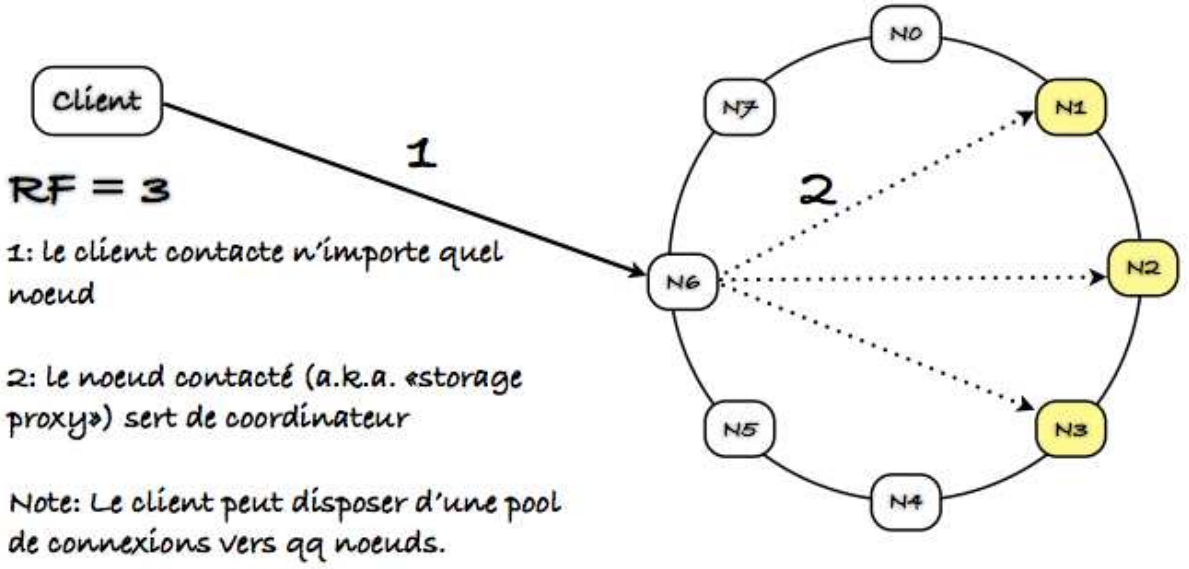


Le client afin d'être sûr que la donnée est effectivement écrite et répliquée, doit déterminer le **tuneable Consistency Level**. Il existe trois niveaux :

- ONE : Il faut que la donnée soit écrite sur au moins un nœud, les réplicas seront faits de manière asynchrone.
- Quorum : La majorité des réplicas ont été écris, les autres seront répliqués de manière asynchrone.
- ALL : Tous les réplicas doivent être écrits.

2. Lecture

Pour lire des données sur un cluster Cassandra, le client se connecte sur le premier nœud qui répond à sa demande.



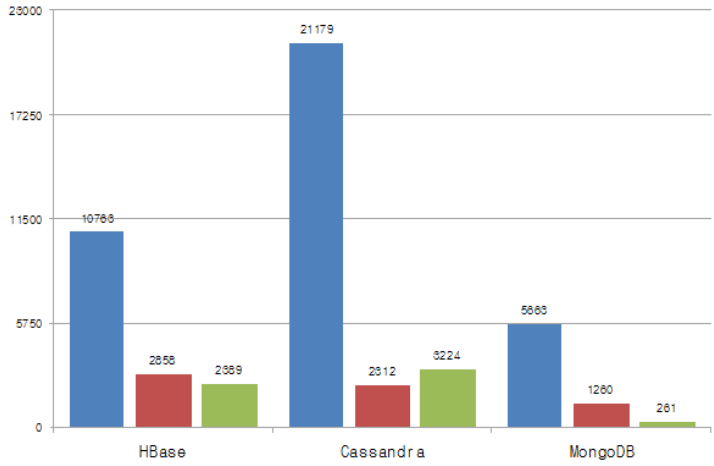
Afin de s'assurer que la donnée qu'il lit est la plus récente, il est nécessaire lors de la requête de mettre en place un Tuneable Consistency Level. De même que pour l'écriture, il existe trois niveaux :

- ONE : La donnée d'un réplica est retournée (peut ne pas être la dernière mis à jour)
- QUORUM : Retourne la donnée la plus récente parmi la majorité des réplicas
- ALL : Retourne la donnée la plus récente parmi tous les réplicas

Afin d'être sûr ou presque d'avoir toujours la dernière données, il faut écrire avec une constance = QUORUM et de même pour la lecture. Ceci peut par contre ralentir sensiblement la solution Cassandra.

3. Performance

Le test effectué est réalisé sur 50 Million d'enregistrements de 1Ko soit 50 Go de données. Les résultats sont en opération par secondes.



Nous voyons donc ici que Cassandra est beaucoup plus rapide que ces homologues NoSQL en écriture et en mise à jour. Mais que la lecture est équivalente à celle de HBASE.

E. Les avantages et inconvénients

1. Avantages

En ce qui concerne les avantages de cette solution, ils sont nombreux et ouvrent de bonnes perspectives.

En ce qui concerne l'architecture des données, le fait que cette technologie fasse fie des contraintes d'intégrité permet qu'il n'y ait pas de jointure, elles sont donc à faire du côté client. De ce fait, le serveur dispose de toute ça puissance pour répondre au besoin. Les calculs et agrégations sont fait coter client.

Le principal problème des systèmes de gestion de bases de données relationnelles classique réside dans le fait que ces systèmes sont installés sur des machines physiques uniques. La plupart du temps les entreprises ont plusieurs serveurs pouvant récupérer la charge en cas de défaillance. Cassandra est une solution distribuée généralement de grande ampleur. Du fait de son architecture, la défaillance de l'un de ces nœuds est complètement transparente dans la cadre de son utilisation. En effet dans la mesure où la charge est répartie sur l'ensemble des nœuds constituant le serveur, la perte d'un nœud est largement compensée par le nombre de nœud actifs. Les problèmes liés aux défaillances hardware passent donc au second plan.

Au niveau des connexions des utilisateurs, le fait que le serveur soit réparti sur plusieurs nœuds, lors de la connexion d'un utilisateur sur le serveur, celle-ci est établie avec le nœud qui répond en premier en fonction de la latence réseaux. Il y a donc plusieurs points de connexion possibles. Cet aspect est intéressant puisqu'il efface les points individuels de défaillances (en anglais, single point of faillure).

Avec cette technologie qu'est le NoSQL, il n'existe plus d'intégrité référentielle, plus de relation, plus de schéma. Les données sont regroupées de façon similaire aux bases de données classiques mais elles sont complètement libres d'utilisations ou de manipulations.

Dans un souci de performance, le nombre de nœuds joue un rôle important aussi bien en termes de temps de réponse que de capacité de stockage d'information. Chaque nœud représente une capacité de stockage supplémentaire, de plus chacun d'eux représente un nouveau point d'entrée sur le serveur. La « scalabilité » de la solution Cassandra est dite linéaire, c'est-à-dire que le serveur sera deux fois plus performant si l'on multiplie le nombre de nœuds par deux.

Voici un tableau comparatif permettant de voir ces informations.

	48 nodes	96 nodes	144 nodes	288 nodes
Writes Capacity	174373 w/s	366828 w/s	537172 w/s	1,099,837 w/s
Storage Capacity	12.8 TB	25.6 TB	38.4 TB	76.8 TB
Nodes Cost/hr	$32.64	$65.28	$97.92	$195.84
Test Driver Instances	10	20	30	60
Test Driver Cost/hr	$20.00	$40.00	$60.00	$120.00
Cross AZ Traffic	5 TB/hr	10 TB/hr	15 TB/hr	30[1] TB/hr
Traffic Cost/10min	$8.33	$16.66	$25.00	$50.00
Setup Duration	15 minutes	22 minutes	31 minutes	66[2] minutes
AWS Billed Duration	1hr	1hr	1 hr	2 hr
Total Test Cost	$60.97	$121.94	$182.92	$561.68

Un nœud peut être ajouté ou être supprimé du cluster de nœuds sans interruption de service et sans le moindre impact sur le fonctionnement et la disponibilité globale des données. Ces interventions sont donc, tout a fait imperceptible aux niveaux des utilisateurs même s'ils sont connectés pendant l'intervention.

Le principal avantage et le but même du mouvement était de répondre à un problème simple : La demande de montée en charge de vastes systèmes. Aujourd'hui, la montée en charge des technologies NoSQL est théoriquement illimitée. Elle dépend essentiellement du nombre de nœuds constituant le cluster et donc cette capacité à monter en charge est variable d'un système à l'autre.

De plus, il n'y a pas de système de « lock » ni de transactions sur les données. Lors de manipulation de données, avec les systèmes de gestion de bases de données relationnelles, l'accès concurrentiel aux données est géré de sorte que le premier utilisateur à utiliser la donnée bloque l'accès à celle-ci aux autres utilisateurs pour ne pas impacter l'intégrité de la donnée. Avec Cassandra, l'accès aux données est toujours garanti. Cela pose toutefois un souci de cohérence de données.

2. Inconvénients

Certaines des forces relevées dans le paragraphe précédent induisent des faiblesses de surcroit.

Cette technologie ne gère pas les contraintes d'intégrité, c'est un choix pour permettre plus de flexibilité au niveau de l'interrogation de données. Des questions apparaissent naturellement, « Mais comment sont organisées les données ?, Comment gérer les doublons ?, Comment identifier simplement un ensemble de données ?, Quelle est le schéma de base de données ? ». Ces réponses doivent être prises en compte lors des développements applicatifs. De plus, sans contrainte d'intégrité, il est impossible d'annuler une transaction une fois la mise à jour ou l'insertion effectuée. En effet, il n'existe plus l'étape de vérification de passage de ladite transaction.

Cette solution nécessite une architecture hardware importante. Cette importance dépend souvent de l'utilisation pour laquelle cette solution est choisie. Le déploiement d'une telle solution induit des coûts très importants afin de garantir une solution pérenne. Chaque nœud nécessite d'être installé sur un serveur indépendant.

A la différence des systèmes de gestion de bases de données, Cassandra ne dispose pas de gestion de cache. Les requêtes sont toutes rejouées. Le système de cache permet de placer le résultat des requêtes souvent utilisées en mémoire, afin de ne pas provoquer trop d'entrées/sorties disque dur inutiles.

L'utilisation du NoSQL est rare aujourd'hui à cause d'un problème relevé par les grands éditeurs de solutions. En effet, l'aspect sécurité des données est géré avant la connexion d'un utilisateur. Cependant, une fois connecté, l'utilisateur à la possibilité de voir et de modifier l'ensemble des données du serveur.

VI. Mise en œuvre de cette solution

A. Pré requis

La mise en œuvre d'une telle solution nécessite de faire une étude du système existant ainsi que du système cible souhaité. Le périmètre et l'utilisation de la future solution doivent être définis en amont. Les besoins en investissement matériels ou en cout de déploiement vont être proportionnels à la demande en termes de capacité dont disposera le futur serveur à faire face à la montée en charge.

1. Hardware

Nous avons vu précédemment que les performances sont directement liées au nombre de nœuds dont la solution dispose en fonction de la demande de montée en charge. Un serveur Cassandra est installé sur au moins trois nœuds.

a. CPU

La performance du serveur Cassandra sera en premier lieu lié au nombre de processeurs avant d'être en relation avec la mémoire dont il dispose. Cassandra est adaptée à l'utilisation d'un grand nombre de processeurs. Dans la mesure où une solution Cassandra est installée sur au moins trois serveurs, il disposera de plusieurs processeurs qui seront disponibles pour répondre aux différentes sollicitations des utilisateurs. Il faut comprendre ici que ce n'est pas la puissance d'un processeur qui est pris en compte mais belle et bien la puissance fournie par l'ensemble de tous les processeurs présents sur les nœuds. Cette puissance est fixée en fonction du nombre de nœuds et de la demande face à la montée en charge du système.

b. Mémoire

La performance est aussi liée en grande partie à la mémoire dont dispose chaque nœud du serveur. En effet, plus il y aura de mémoire plus les requêtes seront rapides à traiter. La charge étant répartie, chaque nœud disposera d'un volume moyen qui sera fonction de l'espace disque disponible sur ledit nœud.

c. *Espace disque*

Chaque nœud disposera d'au moins un disque dur dont le volume est variable. Cette part de variable est fonction de la taille de la base de données et du besoin estimé en termes de volume de données à traiter par jour. L'espace disque du serveur correspond à l'ensemble des espaces disque présent sur les nœuds. Si un serveur est composé de 5 nœuds dont chacun dispose de 20Go, on dira que le serveur dispose de 100Go au total.

2. Software

Une installation de Cassandra sera nécessaire autant de fois qu'il y aura de nœuds déployés.

Par ailleurs, Le fait de cette solution ne dispose pas de gestion de cache intégré ni de contrainte d'intégrité, ces aspect doivent avoir été pensés et mis en place au niveau des applications qui viendraient se baser sur ce serveur. Cela nécessite donc une réflexion supplémentaire au niveau de la conception et du développement des applications.

B. Installation / configuration

Une fois que les questions de nombre de nœuds, de la taille de la base de données à terme et du volume de données sont traitées, l'installation de la solution peut alors débuter. Chaque nœud doit disposer d'au moins 4G de mémoire RAM, un processeur de 3Ghz et d'un disque dur dont le volume est suffisent en fonction de la taille de la base de données. L'application Cassandra est disponible en téléchargement libre sur le site : http://cassandra.apache.org/ . L'application doit être installée sur chacun des nœuds. L'installation d'un serveur s'effectue principalement sur des systèmes linux. Toutefois il est possible de l'installer sur des systèmes Windows. Une différence est à noter en fonction du système, elle porte sur le paramétrage de l'application que nous allons voir dans le paragraphe suivant. L'installation s'effectue très simplement. Dans un premier il faut télécharger l'application qui sera archivée. Ensuite, il faut extraire les fichiers qui viennent d'être téléchargés. Parmi ces fichiers, se trouve le fichier « conf/storage-conf.xml ». Il s'agit ici de configurer les différents répertoires utiles au bon fonctionnement de Cassandra.

Sous UNIX :

```
<CommitLogDirectory> /var/lib/cassandra/commitlog </ CommitLogDirectory>
<DataFileDirectories>
      <DataFileDirectory> /var/lib/cassandra/data </ DataFileDirectory>
</ DataFileDirectories>
<CalloutLocation> /var/lib/cassandra/callouts </ CalloutLocation>
<BootstrapFileDirectory> /var/lib/cassandra/bootstrap </ BootstrapFileDirectory>
<StagingFileDirectory> /var/lib/cassandra/staging </ StagingFileDirectory>
```

Sous Windows :

```
<CommitLogDirectory> D:/cassandra/data/commitlog </ CommitLogDirectory>
 <DataFileDirectories>
        <DataFileDirectory> D:/cassandra/data/data </ DataFileDirectory>
 </ DataFileDirectories>
 <CalloutLocation> D:/cassandra/data/callouts </ CalloutLocation>
 <BootstrapFileDirectory> D:/cassandra/data/bootstrap </ BootstrapFileDirectory>
 <StagingFileDirectory> D:/cassandra/data/staging </ StagingFileDirectory>
```

Voici deux exemples de configuration possibles.

Une fois que la configuration adéquate est réalisée sur chacun des nœuds, il est possible de tester l'installation en exécutant Cassandra.

Sous Unix, il faut se placer dans le répertoire Cassandra courant et taper « ./Cassandra ». L'application va donc se lancer sur le nœud.

Sous Windows, dans le répertoire d'installation dans le répertoire « bin », se trouve un fichier cassandra.bat. Ce fichier lance l'application Cassandra sur le nœud concerné.

C. Cas d'utilisation

Aujourd'hui, le NoSQL est une technologie au stade de l'émergence. Le NoSQL est mis en œuvre dans le cadre de très gros sites internet tels que facebook, LinkedIn ou twitter. Ces derniers ont des besoins énormes en stockage et traitement des données. Il n'y a pas ou que très peu d'accès concurrentiels aux données en écriture ou modification. Les réseaux sociaux ont besoins de gérer une très grosse montée en charge dû aux grands nombres d'utilisateurs et donc de requêtes simultanées.

Cassandra permet de disposer de très grandes performances en écriture et en modification, et dispose d'aussi bonne statistique que ces concurrents en lecture.

Cette solution n'est pas utilisée dans un cadre applicatif à cause de la non cohérence des données en temps réel sur l'ensemble des nœuds. Lorsqu'une donnée est mise à jour sur un nœud, il faut un moment avant que la donnée soit dupliquée sur plusieurs nœuds et qu'elle soit accessible depuis n'importe lequel d'entre eux. Lorsqu'une donnée est modifiée plusieurs fois de suite, celle qui est utilisée est celle qui porte la date de modification la plus récente. Chaque donnée est liée à une date sous forme de timestamp, ce qui permet de savoir sur quelle donnée se baser dans le cas ou la donnée est modifiée plusieurs fois de suite.

Cet aspect ne pose pas de problème dans le cas d'une utilisation pour un réseau social, l'enjeu au niveau de la cohérence des données en temps réel passe au second plan par rapport à la disponibilité ou au très grand volume de données à traiter en simultané.

Ce dernier point représente l'enjeu principal de la technologie NoSQL.

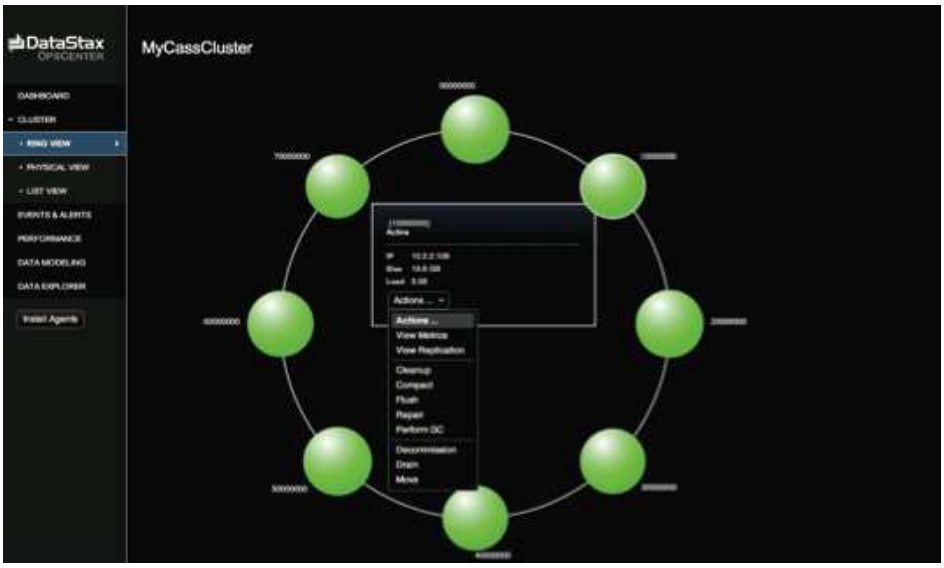
VII. Préconisations

A. Software

1. DataStax OPS Center

DataStax OPS Center est la première solution de gestion, de surveillance et exploitation des clusters d'entreprises de bases de données Cassandra. Avec OPS Center, les utilisateurs de Cassandra ont une plateforme pour gérer les applications en temps réel, ainsi que des rapports d'analyse et de business intelligence. DataStax OPS Center fournit aux utilisateurs des visualisations de leurs clusters, une gestion globale et une capacité de configuration des clusters. L'outil permet aussi de suivre la santé et les performances de l'ensemble des serveurs du cluster Cassandra. Le client OPS Center est accessible via un simple navigateur Web.

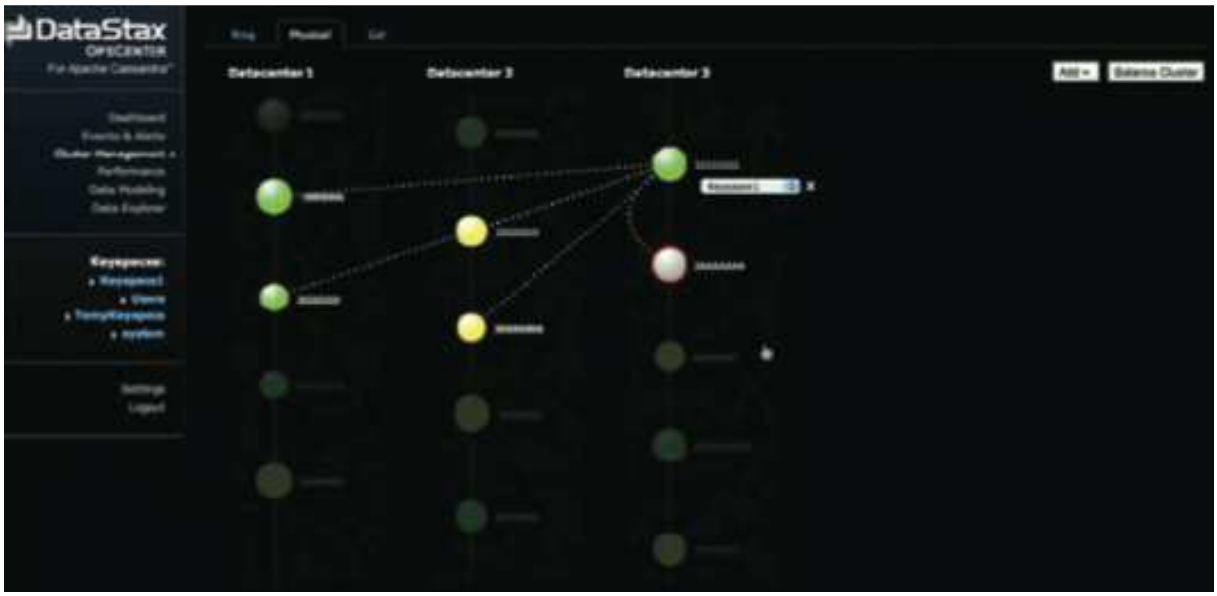
Sur cette précédente capture d'écran, nous voyons le tableau de bord du cluster, nous y retrouvons le suivi des alertes et des événements ainsi que des statistiques sur l'utilisation du cluster (espace disque, nombre d'opérations de lecture et d'écriture à la seconde ou encore la latence du cluster).



Nous voyons ici l'ensemble des nœuds du cluster qui forme l'anneau de serveurs de Cassandra.

De plus, DataStax OPS Center permet un grand nombre d'opération sur le cluster. Tel que l'installation distante de l'agent OPS Center, cet agent permettant de faire les interactions entre le serveur et OPS Center. De plus, l'OPS Center permet de gérer et d'explorer les données du cluster, nous pouvons alors ajouter, supprimer et mettre à jour les données directement via l'interface Web d'OPS Center.

Pour finir, OPS Center gère entièrement les clusters répartit sur plusieurs Datacenter, il est alors possible (dans la version entreprise) de faire du rebalancing automatic entre les différents serveurs.



DataStax OPS Center est une solution gratuite qu'il est possible de télécharger sur le site de DATSTAX.

L'installation d'une telle solution est préférable sur des systèmes Unix, linux. Ces systèmes sont connus pour leur stabilité et consomment beaucoup moins de ressources. Ils permettront de faire des économies de mémoire RAM et de laisser plus de ressources pour les nœuds installés.

B. Hardware

Le fait de devoir disposer d'un nombre important de serveurs est non négligeable en termes de coûts et de maintenance. Afin de minimiser les coûts provoqués par les besoins en matériels, la solution de virtualisation des nœuds est bien adaptée.

Pour une solution devant disposer de plus de 10 nœuds, il est possible de virtualiser l'ensemble des nœuds. Il faut tout de même prendre en compte le fait qu'il faille plusieurs serveurs physiques. Trois serveurs physiques sont indiqués pour un ensemble de 10 nœuds Cassandra. Chacun de ces nœuds doit disposer d'un minimum de 4G de mémoire RAM. Les serveurs physiques devront disposer d'au moins 15 à 20G de mémoire RAM.

En termes de disques durs il est préférable de prévoir deux disques durs par nœuds, l'un pour les données de la base de données, l'autre pour les fichiers de log qui assureront un suivi et une base en cas de reprise d'activité ou en cas de reprise d'historique. Le fait qu'il y ait deux disques durs va permettre de garantir la non perte de données. Si l'un des deux est perdu, une intervention est possible rapidement du fait de la virtualisation. De plus, si le disque dur de données est perdu, il suffira de rejouer les logs de l'autre disque dur.

Au niveau de la configuration, il est important de configurer les nœuds d'un serveur physique comme des nœuds « voisins ». Avec un mode d'écriture en QUORUM, la donnée sera répliquée sur la majorité des nœuds. De ce fait, dans le cas où un serveur physique tombe, les données ne seront pas perdues et la charge sera répartie parmi les nœuds disponibles. L'utilisation d'OPS Center permettra de prévenir ces cas.

VIII. Bilan/ouverture

La technologie NoSQL a été crée dans le but de répondre à deux besoins. Le premier besoin exprimé a été les difficultés qu'ont les systèmes de gestion de bases de données face à la montée en charge dû aux besoins toujours grandissant d'internet, le second besoin était une demande de flexibilité par rapport aux contraintes d'intégrité induites par l'utilisation de systèmes de gestion de bases de données relationnelles. Des solutions ont été fournies pour répondre à ces demandes. En contrepartie, ces solutions paraissent n'être adaptées, en l'état actuel, qu'au monde du WEB.

La technologie reste jeune et est encore en cours de développement. Le concept a été repris par de grands éditeurs qui travaillent à combler les lacunes des solutions existantes.

Le NoSQL est-il une solution complémentaire ou alternative aux systèmes de gestion de bases de données relationnelles classiques ? Sera-t-il, à terme, adapté à une utilisation dans le cadre applicatif ?