Synthèse d'étude et projets d'intergiciels

Base NOSQL

Mathieu ROGER
octera [AT] octera [DOT] info

Résumé

Devant le besoin grandissant en performance et en disponibilité des services/sites possédant un fort trafic, un point souvent bloquant est la base de données. Les bases de données relationnelles atteignent vite leurs limites et rajouter des serveurs n'augmente plus assez les performances. Suite à ce problème, de nouvelles technologies ont émergé tels que les bases de données NoSQL, celles-ci changent radicalement l'architecture des bases de données que nous avons l'habitude de voir et permettent ainsi d'augmenter les performances et la disponibilité des services, il y a bien évidemment des « mais », la réponse parfaite n'existe pas.

Abstract

Faced with the growing need for performance and availability of services / websites with heavy traffic, a often blocking point is the database. Relational databases quickly reach their limits and add more servers don't increases the performance. Thanks to this problems, many new technologies have emerged such as database NoSQL, it radically changes the architecture of the databases that we used to see and thus increases the performance and availability of services. Of course, there are "but", there is no perfect answer.

Table des matières

Limites du modèle relationnel

Les bases de données existent maintenant depuis environ 50 ans et le modèle relationnel depuis environ 40 ans, ce modèle bien que très puissant, a des limites que certains services/sites, tels que Google, Facebook, etc, ont atteintes depuis longtemps. En effet ce genre de site possède plusieurs millions voir milliard d'entrées dans leurs bases de données et tout autant de visites journalières en conséquence une seule machine ne peut pas gérer la base de données, de plus pour des raisons de fiabilité ces bases de données sont dupliquées pour que le service ne soit pas interrompu en cas de panne. La méthode consiste donc à rajouter des serveurs pour dupliquer les données et ainsi augmenter les performances et résister aux pannes.

Seulement, dû aux propriétés fondamentales sur lesquels une base de données repose cette approche a ses limites : une base de données relationnelle est construite sur les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité), ses propriétés bien que nécessaires à la logique du relationnel nuisent fortement aux performances surtout la propriété de cohérence. En effet, la cohérence est très difficile à mettre en place dans le cadre de plusieurs serveurs, car pour que celle-ci soit respectée tous les serveurs doivent être des miroirs les uns des autres, de ce fait deux problèmes apparaissent :

- Le coût en stockage est énorme car chaque donnée est présente sur chaque serveur.
- Le coût d'insertion/modification/suppression est très grand, car on ne peut valider une transaction que si on est certains qu'elle a été effectuée sur tous les serveurs et le système fait patienter l'utilisateur durant ce temps.

De plus certains types de requêtes ne sont pas du tout optimisés dans ce type de base de données : imaginons une table contenant toutes les personnes vivant en France, soit plus de 60 millions, les données dans une base de données relationnel classique sont stockées par lignes, ainsi si on effectue une requête pour extraire tous les noms des personnes ayant moins de 15 ans, il faudra parcourir les 60 millions d'entrées et accéder au champ âge, or dû à la manière dont sont stockées les données cette informations n'est pas contigüe en mémoire de ce fait ont perd en performances car il faut du temps pour se placer sur la bonne donnée.

A cause de ces limitations, de nouvelles technologies ont été créées pour répondre à ces besoins, c'est ainsi qu'a été inventé d'autres architectures, celles-ci se nomment dans leur ensemble « bases NoSQL » pour « Not Only SQL »

Les bases NoSQL

Les bases NoSQL répondent aux besoins que nous avons vus précédemment. Comme il n'existe pas de solution miracle, il existe de nombreux types de bases de données spécifiques à des besoins différents. En règle générale ces bases de données respectent le théorème de CAP d'Eric Brewer, ce théorème consiste en trois notions :

- Consistance (consistency /C) : Tous les clients voient la même vue même lorsqu'il y a des mises-à-jour.
- Haute disponibilité (availability /A) : L'ensemble des clients peuvent trouver des données répliquées, même lorsqu'une panne survient.
- Tolérant à la partition (partition-tolerance /P) : Le système est tolérant au partitionnement.

Ce théorème nous apprend aussi qu'il est impossible de respecter ces trois propriétés simultanément, nous ne pouvons en respecter seulement deux à la fois. Dans la majorité des systèmes se sont les propriétés A et P qui sont respectées. En effet dans beaucoup d'applications la consistance peut ne pas être un problème immédiat : si une mise à jour survient, les serveurs n'ont pas tous les mêmes données juste après la mise à jour, mais cette mise à jour va se propager plus ou moins lentement

suivant le type de base de données et les paramètres échéant, donc « au bout d'un moment » les serveurs auront tous effectués cette mise à jour. Prenons l'exemple de Facebook :

- Si la propriétés C n'est pas présente : un de vos amis publie quelque chose, vous ne le voyez pas immédiatement mais au bout de quelques minutes
- Si la propriétés A n'est pas présente : un serveur tombe en panne, la moitié de vos amis ne sont plus accessible et de plus à chaque fois que vous publiez quelque chose votre navigateur met plusieurs secondes à charger la page.
- Si la propriété P n'est pas présente : un serveur tombe en panne, la moitié de vos amis ne sont plus accessible, tout le système est ralenti, et vous ne pouvez plus publier quoi que ce soit.

Pour les entreprises le choix est facile, même dans un monde plus professionnel tel qu'un site marchand, celui-ci préférera que le client commande un produit qui n'est plus en stock au risque de devoir le contacter pour annuler la transaction plutôt que de voir tout le site ralenti et ainsi perdre de nombreux client.

Dû à la perte de consistance des problèmes de mise à jour peuvent se produire, si par exemple un client A modifie une donnée sur un serveur et qu'au même moment un client B modifie la même donnée deux versions différentes de cette données va se propager. Il y a donc une incohérence dans cette donnée, il existe plusieurs solutions à ce problème :

- Soit on prend la version la plus récente (pas forcément évidant du à l'impossibilité de synchronisé véritablement les horloges entres les serveurs)
- Soit le serveur fait remonter aux clients l'erreur pour qu'ils le corrigent eux mêmes

Il existe différents types de base de données NoSQL spécifiques à différents besoins, en voici quelques uns avec le nom des solutions :

- Orientées colonnes
  - HBase
  - Hypertable
  - Cassandra
  - BigTable
- Orientées graphes (Euler)
  - Neo4J
- Orientées clé-valeur
  - Voldemort
  - Dynomite
  - Riak
- Orientées document
  - CouchDB.

Nous allons donc maintenant décrire ces différents types de base de données NoSQL.

Bases orientées colonnes

Les bases de données orientées colonnes ne diffèrent pas beaucoup des bases SQL classique du point de vue architectural, généralement le modèle n'est pas présent pour des questions de performances. La principale différence se situe à la façon dont les données sont stockées en mémoire : dans une base de données relationnelles les données sont stockées par ligne, de ce fait une requête portant sur peu de lignes mais beaucoup de colonnes sera rapide car le système n'aura pas à beaucoup se

déplacer en mémoire, alors que dans une base orientée colonnes les données sont stockées par colonnes, de ce fait le système sera beaucoup plus rapide pour des requêtes portant sur peu de colonnes, il est donc beaucoup plus rapide de parcourir les lignes d'une table.

Bases orientées graphes

Les bases de données orientées graphe permettent de résoudre des problèmes très difficiles voir impossibles à résoudre dans une base de données relationnelle. Prenons exemple sur Facebook, si nous devions stocker dans une base SQL classique les relations entres les personnes la base serait très grosse et donc longue à parcourir pour rechercher les amis d'une personne.

Une base de données orientée graphe stocke les informations sous forme de graphe, elle permet de relier les entités entre elles et de manière optimisée.

La performance de ce type de base de données est très difficile à évaluer car trouver les relations d'une entité est très rapide, mais trouver une entité spécifique si on a aucune connaissance du système est très long, ce système est donc utilisé en parallèle d'une base de données SQL, celle-ci stockera les informations sur les entités et où aller la chercher dans le graphe, alors que la base orientée graphe stockera les relations entres les entités.

Bases orientées clé-valeur

Les bases de données orientées clé-valeur permettent de stocker une valeur, cette valeur peut être de tous type (entier, chaine de caractères, flux binaire, etc.) En revanche les requêtes ne portent que sur la clef associée à cette valeur. Ce système de base de données est conçu pour être très fortement répliqué de manière à augmenter la disponibilité et les performances. La réplication est données est plus ou moins partielle pour trouver un bon compromis entre nombre de serveurs, disponibilité et espace disque.

Bases orientées document

Les bases de données orientées document sont une extension des bases orientées clé-valeur, à la place de stocker une valeur, nous stockons un document. Un document peut contenir plusieurs valeurs et d'autres documents, qui peuvent à leur tour en contenir d'autres et ainsi de suite. Un document peu donc posséder plusieurs niveaux de profondeur. Tous les documents de niveau 0 sont identifiés par une clef et sont regroupés dans une collection.

Ce système est capable de faire des recherches sur les valeurs d'un ou plusieurs documents, le système étant partiellement répliqué, la requête va être envoyée à tous les serveurs simultanément et chacun va effectuer celle-ci et rendre des résultats différents, car toutes les données ne sont pas présentes sur tous les serveurs, ces résultats vont être ré-agrégés pour former le résultat finale à la requête.

Pour conclure les bases de données NoSQL permettent de rendre le système beaucoup plus performant et résistants aux pannes, en revanche comme celles-ci ne permettent pas de cohérence des données, elles ne sont que très rarement utilisées seule : une base de données relationnelle va contenir les informations où une cohérence est vitale et une base de données de type NoSQL va contenir tout le reste.

Les bases de données de type NoSQL étant des technologies encore très récente, il n'y a pas encore de normes qui permettent de définir une architecture type pour tel ou tel type de base de données, ni de syntaxe particulière ; bien que le mouvement tend vers une convergence pour des requêtes basées sur le langage JavaScript.

Références :

http://davidmasclet.gisgraphy.com/post/2010/06/09/10-minutes-pour-comprendre...NoSQL
Article regroupant les grands principes des bases de données NoSQL et une brève description de chacune.

http://en.wikipedia.org/wiki/Nosql
http://en.wikipedia.org/wiki/Graph_database
http://en.wikipedia.org/wiki/Document-oriented_database
http://en.wikipedia.org/wiki/Column-oriented_DBMS
http://en.wikipedia.org/wiki/Relational_database
Diverses informations sur toutes les notions abordées durant la synthèse et la présentation.

http://www.mongodb.org/
Site officiel de MongoDB avec différents articles, tutoriels, documentations sur leur base de données et les bases de données orientées documents.