Concepts de base du NoSQL

16 avril 2018

1 La distribution

L'une des raisons qui ont conduit à la naissance du NoSQL est de pouvoir distribuer les données sur plusieurs machines, afin d'assurer la gestion de données de plus en plus volumineuses, ainsi qu'une montée en charge la plus rapide et indolore possible. L'architecture distribuée est donc au cœur du NoSQL. Il existe deux grandes façons de considérer la distribution des données : la distribution avec ou sans maître. Les moteurs NoSQL utilisés pour gérer de grands volumes de données ont fait l'un ou l'autre choix (voir [1, 2]).

1.1 La distribution avec maître ou centralisée

La distribution avec maître s'appuie sur la présence d'une machine maître, qui conserve la configuration du système, reçoit les requêtes des clients et les redirige vers les machines qui contiennent les données désirées. On écrit sur le maître, qui se charge de répliquer les données sur les esclaves. Ces machines secondaires peuvent, en revanche, être utilisées en lecture, tout dépend du besoin de cohérence de la lecture. En effet, permettre la lecture sur un réplica implique qu'il est possible de lire une donnée obsolète, qui viendrait d'être mise à jour sur le serveur maître, mais pas encore répliquée. La problématique d'un système distribué avec maître est bien entendu la présence d'un SPOF (Single Point Of Failure), c'est-à-dire d'un élément non redondant qui représente un risque de défaillance pour tout le système.

1.2 La distribution sans maître ou décentralisée

Certains moteurs NoSQL se basent sur une architecture sans maître, où théoriquement toute machine a la même importance et les mêmes capacités dans le cluster. L'objectif est de pouvoir assurer un service complet en se connectant à n'importe quelle machine, même si plusieurs membres du cluster tombent.

2 Mecanismes de distribution

Pour bâtir un système distribué fonctionnel, il faut répondre à plusieurs questions : comment diriger les requêtes des clients sur les machines qui contiennent l'information

souhaitée ? Et comment maintenir l'état du système, notamment comment savoir quelles sont les machines qui composent le cluster ? Il faut aussi savoir où et comment distribuer les données, de façon à ce qu'elles soient réparties autant que possible et qu'elles puissent être répliquées sur différents sites. Nous allons aborder quelques concepts et technologies qui répondent à ces questions.

2.1 Le protocole de bavardage

Un protocole de bavardage (gossip protocol) est une méthode de distribution des informations virales basée sur ce qu'on pourrait appeler la théorie des épidémies. En résumé, si une personne est contagieuse, elle va entrer en contact avec des sujets réceptifs qui vont tomber malades et eux-mêmes contaminer d'autres sujets, ce qui a un effet de croissance exponentielle. Les systèmes distribués peuvent utiliser le protocole de bavardage pour répandre des informations, comme la disponibilité ou l'ajout d'un nœud dans le système ou la modification d'une donnée. Dans les moteurs NoSQL à distribution décentralisée et asynchrone, le protocole de bavardage est aussi utilisé pour réconcilier les données de façon à ce que tous les réplicas obtiennent à terme la version la plus actuelle des données.

2.2 Le hachage consistant (consistent hashing)

Bâtir un système distribué implique également de partitionner les données pour les distribuer de la façon la plus équitable possible entre les nœuds. A priori, le moyen le plus logique pour réaliser cela est de se baser sur la clé (primaire) de la donnée (par exemple, l'identifiant du document), et de trouver un moyen, à partir de cette clé, de distribuer les données. Dans un système sans maître, sans coordinateur, il faut aussi que les clients puissent être redirigés sur les machines qui contiennent les données voulues. Une réponse basique à ces problématiques est la distribution des clés avec un modulo. En général, il s'agit d'un hachage de la clé à l'aide d'un algorithme qui assure une bonne distribution du résultat et retourne une valeur entière. Pour illustrer cela, bâtissons un système distribué sur cinq nœuds. Nous voulons distribuer nos données de la façon la plus régulière possible. Notre algorithme d'attribution des données est basé sur un modulo : lorsque nous créons un nouvel enregistrement, nous calculons un modulo 5 par rapport à la valeur de l'identifiant. Le reste de la division par cinq nous retourne le numéro du nœud auquel nous allons attribuer l'enregistrement. Et par exemple, l'enregistrement portant l'identifiant 48 sera envoyé sur le nœud 3. En pratique, cela pose un problème lorsqu'on va souhaiter ajouter un nœud. Nous devons modifier notre modulo pour qu'il corresponde au nouveau nombre de nœuds, ce qui implique une réorganisation des données sur tous les nœuds. C'est un défaut majeur, et sur un système de grande envergure, c'est même pratiquement inenvisageable. C'est pour éviter cette problématique que la technique du consistent hashing a été développée.

L'idée du hachage consistant est d'attribuer des rangées de valeurs pour chaque nœud. Conceptuellement, le système distribué est vu comme un anneau (hash ring, anneau de hachage), où chaque nœud occupe une position sur l'anneau. Chaque nœud se voit attribuer une valeur de hachage, qui définit la rangée de valeurs qui est hébergée. L'algorithme

de hachage calcule un hash code à partir de la clé, trouve le nœud dont la valeur est immédiatement supérieure et y stocke la donnée. Un nœud contient donc toutes les clés inférieures à sa valeur de rangée et supérieures à la valeur du nœud précédent. Lorsqu'un nœud est ajouté à l'anneau, il prend une valeur de rangée, et donc scinde une rangée existante. L'attribution des nœuds ne sera donc troublée que pour une seule rangée, et un nombre peu important de données sera déplacé.

Toutefois, il ne s'agit pas forcément de la solution la plus performante dans tous les cas. Une autre méthode existe, qui consiste à découper les données en nœuds virtuels, et à gérer la réorganisation des données lors d'un redimensionnement du cluster par nœud virtuel.

2.3 Les VNodes

Le hachage consistant présente deux problèmes. Tout d'abord, l'ajout d'un nœud va couper les données d'un seul autre nœud, ce qui ne contribue pas à l'équilibre général du cluster. Tout au plus, cela peut rééquilibrer un seul nœud, ou réduire la charge relative de ce nœud et du nouveau par rapport aux autres. L'idéal serait plutôt une méthode qui maintienne un équilibre général. Par ailleurs, l'échange de données entre les deux machines va fortement charger le nœud coupé, qui sera donc sollicité en exclusivité, et le trafic réseau entre ces deux machines subira un pic. Il faudrait un moyen qui, sans provoquer une migration aussi importante, permette aussi de distribuer la charge de l'alimentation du nouveau nœud. Cette technique, c'est celle des nœuds virtuels.

La base de données va être découpée, non pas en segments qui correspondent à des machines réelles, mais en beaucoup plus de blocs, qui seront répartis sur les nœuds. Chaque bloc représentera donc une unité et chaque nœud physique va héberger un nombre proche de blocs. Lors de l'ajout d'un nouveau nœud, chaque machine va choisir un certain nombre de ses blocs pour les envoyer sur le nouveau membre. Cela permet d'assurer un bien meilleur équilibre et de répartir la charge réseau lors de la réorganisation du cluster. Cela ne représentera pas de charge inutile, car seul le bon nombre de blocs sera déplacé.

3 La réplication et la consistence

3.1 La réplication

La réplication c'est une copie des données qui sert avant tout à assurer la redondance, donc la protection contre la perte de données. Les systèmes NoSQL utilisent la réplication pour atteindre plusieurs objectifs.

— Disponibilité. La réplication permet d'assurer la disponibilité constante du système. En cas de panne d'un serveur, d'un noeud ou d'un disque, la tâche effectuée par le composant défectueux peut immédiatement être prise en charge par un autre composant. Cette technique de reprise sur panne instantanée et automatique (failover) est un atout essentiel pour assurer la stabilité d'un système pouvant comprendre

des milliers de noeuds.

— Scalabilité (lecture). Si une donnée est disponible sur plusieurs machines, il devient possible de distribuer les requêtes (en lecture) sur ces machines.

— Scalabilité (écriture). Enfin, on peut penser à distribuer aussi les requêtes en écriture, mais là on se retrouve face à de délicats problèmes potentiels d'écritures concurrentes et de réconciliation.

3.2 La cohérence finale (eventual consistency)

La réponse à la cohérence dans un système distribué, telle qu'elle se présente dans les moteurs NoSQL à distribution asynchrone, s'appelle la cohérence finale (eventual consistency). La cohérence finale laisse le système ouvert, permet la lecture et s'occupe au fur et à mesure de diffuser les modifications sur tous les nœuds, en mettant en place des outils de réconciliation en cas d'incohérence.

4 Le théorème CAP

Le théorème CAP de Eric Brewer est une abréviation de trois propriétés concernant la distribution des données. L'énoncé de ce théorème est le suivant. «*Les bases de données distribuées ne peuvent satisfaire à un instant donné que deux des trois critères suivants : la cohérence, la disponibilité et la résistance au partitionement.*»

— Consistency (cohérence) : tous les nœuds du système voient exactement les mêmes données au même moment ;

— Availability (disponibilité) : la perte d'un noeud n'empêche pas le système de fonctionner et de servir l'intégralité des données ;

— Partition tolerance (résistance au partitionement) : chaque noeud doit pouvoir fonctionner de manière autonome. Cela signifie qu'en cas de perte d'un certain nombre de noeuds ou de communication entre certains noeuds, chaque noeud restant doit permettre un accès au moins aux données qu'il contient.

Références

[1] Bruchez, R. : Les bases de données NoSQL et le Big Data : Comprendre et mettre en oeuvre. Eyrolles (2015)

[2] Harrison, G. : Next Generation Databases : NoSQL, NewSQL, and Big Data : What every professional needs to know about the future of databases in a world of NoSQL and Big Data. Apress (2016)