Transparency-One / Multi DC architecture

Le contexte

Suite à la phase béta, on s'aperçoit que beaucoup de supply chain, à partir du niveau 3, pointent vers la Chine. Ainsi la Chine est un bon candidat pour être un client de l'application. Or les temps de réponses ne sont pas acceptable à l'état actuel (la résolution DNS prend plusieurs secondes).

Une discussion est ouverte avec Microsoft Azur et China Telecom, pour avoir un accès raisonnable à l'application.

De plus il est difficile d'implanter l'application en Chine, puisque que pour cela il faut réaliser des accords, avoir une filiale Chinoise.

Les différentes solutions de clustering

Depuis la version 3.1, Neo4j dispose de deux solutions de clustering :

- le clustering haute disponibilité
- le clustering core-edge

Le clustering HA

C'est le mode actuel de clustering de Neo4j. Il est constitué à minima de 3 noeuds, avec forcement un unique noeud master et des noeuds slaves, et chaque noeud possède l'intégralité des données de la base de données (Neo4j ne fait de sharding sur les données).

On peut envoyer des requêtes de lecture/écriture sur tous les noeuds, toutefois en ce qui concerne les requêtes d'écritures, celles-ci sont renvoyées au noeud master: c'est lui qui est le garant de la cohérence des données (rappel : Neo4j est une base ACID). Ainsi il n'y qu'un seul noeud qui traite les écritures.

La synchronisation entre les noeuds est faite selon deux principes :

- **le push :** lorsqu'une transaction est validée par le master, celle-ci peut être poussée à plusieurs noeuds slaves (configurable via le push_factor). Ce push est fait de manière asynchrone en phase de post-commit.
- **le pull :** tous les slaves viennent se connecter au master de manière régulière (toutes les X secondes) pour venir se mettre à jour

En cas de perte du noeud master, une élection est faite entre les noeuds restant pour élire le nouveau noeud master. Bien que cette opération soit rapide, ceci peut conduire à quelques secondes d'indisponibilité en écriture.

Les inconvénients de ce type d'architecture sont :

- il faut une bonne communication intra-cluster, pour éviter qu'un noeud soit suspecté d'être down

- il est possible d'avoir deux masters si une coupure réseaux coupe en deux parties le cluster. Ceci peut conclure à avoir des branch data, c'est à dire deux data-stores qui ont évolués en parallèle.

Le causal clustering

C'est le nouveau mode de clustering de Neo4j, disponible à partir de la version 3.1.

Dans ce type d'architecture il y a deux types de serveurs :

- les cores : ce sont les serveurs actifs sur lesquels ont peut faire du read/writes. Chaque transaction doit avoir un consensus de noeuds core pour être validée.
- les edges : ce sont des serveurs passifs sur lesquels on peut faire uniquement du read.

La synchronisation entre les noeuds core est basée sur le protocole de communication **Raft** qui permet de partager entre ces serveurs, un fichier (dans notre cas le fichier de transaction) de manière transactionnelle. La synchronisation entre les noeuds edge est réalisée de manière asynchrone (pull).

Les avantages de ce type d'architecture sont :

- il ne peut plus y avoir de branch data
- les noeuds edges sont très tolérants aux latences réseaux
- l'ajout de noeud de type edge permet de scaler en lecture, sans aucun impact sur les noeuds cores (nous avons fait des tests avec plusieurs centaines de noeuds).
- la possibilité de faire du **read your own writes** (via la fonctionnalité de markbook)

Les questions posées

Est-ce Core-edge est compatible avec les unmanaged extension ?

Oui l'architecture est compatible, toutefois la fonctionnalité de markbook (pour le read your own writes) ne sera pas disponible. De plus vu les unmanaged extensions fonctionnent en HTTP, vous ne profiterez pas des nouveaux drivers Neo4j qui gèrent le routing des requêtes vers les serveurs Neo4j en fonction du type des sessions (read only, write, ou read your own writes). Ceci est uniquement disponible avec bolt.

Est-ce qu'on peut faire du read-only sur les edges si tous les noeuds cores sont inaccessibles ?

Oui

Est-ce qu'il existe des endpoints permettant de connaître du cluster ?

Oui, il existe deux solutions pour connaître la typologie du cluster :

- soit via une procédure cypher : ``
- soit via des endpoints permettant de savoir si un noeud est de type core ou edge

Ainsi il est toujours possible d'utiliser HAProxy comme load-balancer.

Est-il possible dans le routing des drivers, de mettre une préférence ?

Ici, il s'agit en fait de pouvoir définir un préférence locale au driver d'une application, pour prendre l'information au plus près possible de l'application. Ainsi une application aux USA, se connecterait à serveur edge situé aux USA.

Ceci n'existe pas encore dans la version actuelle, mais est planifié dans la version 3.2.

Quelle solution choisir, et pour quel impact ?

Dans le but de faire du multi-datacenter afin de servir la donnée au plus proche de vos clients, je recommande l'architecture core-edge qui a justement été développé pour ce cas d'usage. Toutefois, cette architecture est en version 1.0, et comme nous l'avons vu, il faudra sans doute attendre la version 3.2 pour pouvoir pousser cette architecture en production.

De plus afin de profiter des nouveaux drivers, et de la fonctionnalité read your own writes, il faudrait effectuer une migration de l'unmanaged extension en procedure.

Refonte de l'architecture

Suite à nos discussions sur l'architecture multi data-center, nous avons aussi discuté de l'architecture applicatif.

Pour arriver jusqu'à Neo4j, il y a plusieurs couches applicatives à traverser :

- l'application en C#
- une application en Java
- l'extension Neo4j

A chaque fois qu'on traverse une de ces couches, cela implique un temps réseaux, mais aussi de la sérialisation/désarialisation en JSon/Bean. Or il y a très peu de traitement métier, sauf au niveau de l'application C# qui permet de réalise simultanément (ou non) des requêtes Neo4j ou ElasticSearch.

Ainsi dans le cadre d'une migration de l'extension Neo4j en procédure, il pourrait être intéressant de penser à faire une fusion de ces applications.

Fonctionnement du causal clustering

Analyse des tests

Description

Le test effectué a été réalisé avec les éléments suivants :

- 3 serveur score en Europe
- 1 replica aux USA
- 1 replica en Asie

Les résultats

En lecture a partir de l'europe, le driver choisit un replica, et donc ne va pas en local (ie en Europe sur un core).

Lors de l'utilisation du bookmark pour récupérer un read, les temps sont de l'ordre de 1,5 secondes. Est-ce que le read se fait sur un replica et donc on attend que le noeud soit a jour ? Ou est ce que cela se fait un core ?

Les réponses

Le fonctionnement des drivers est le suivant (pour l'instant) :

- toutes les actions d'écriture se font sur les replicas
- toutes les actions d'écriture se font sur les cores
- le système de routing est un simple round-robin

Ainsi :

- les requêtes en Europe vont forcément sur un autre continent.
- a cause du round-robin, les requêtes de read ne vont pas sur les serveurs locaux.
- le bookmark réalise l'écriture sur un core, puis attend qu'un replica soit a jour pour le read, d'où la latence.

Les futures versions des drivers, (la 1.3 sortira en avril 2017) viendront avec les fonctionnalités suivantes :

- ajout d'une configuration permettant de lire sur un serveur core, ce qui améliora le cas des requêtes read en Europe et le problème des bookmarks.
- possibilité de configurer le routage via un système de préférence, ou d'implémenter le sien. Ainsi les requête d'Asie iront sur les serveurs en Asie.

En ce qui concerne la gestion des locks, ceux-ci sont gérés par le core `leader`, c'est lui le garant de la cohérence des données.

Pour rappel, il y a deux types de serveur core : un `leader` et des `followers`. Toutes les requêtes d'écriture s'effectue sur le `Leader`, c'est lui le garant de la cohérence des données. Les autres noeuds core (les `followers`), sont la pour établir le consensus de la validité de la requête, ce qui évite les branch-data qu'on avait avec le mode HA. De plus le principe d'élection du Leader est beaucoup plus rapide qu'avec le HA.

Après discution avec notre product manager, la version actuelle du `Causal Clsutering` n'est pas prête pour de la production en multi datacenter, mais c'est bien vers quoi on va. De plus, un breaking change va avoir lieu dans les drivers entre la version 1.1 et la version 1.2.

OGM et `save` complet ou delta ?

D'apres la documentation, OGM fait du delta : https://neo4j.com/docs/ogm-manual/current/reference/#reference:session:persisting-entities

```
1 Upon invocation of save() with an entity, it checks the given object graph
  for changes compared with the data that was loaded from the database. The
  differences are used to construct a Cypher query that persists the deltas
  to Neo4j before repopulating it's state based on the response from the
  database server.
```

Ainsi je ne vois pas le pourquoi de la surcharge de vos objets. Quelle verion d'OGM utilisez-vous ?