HEIG-VD

Rapport sur RavenDB

Distribution, réplication et sharding

Dunant Guillaume & Junod Arthur
23/11/2024

Table des matières

Introduction

RavenDB est une base de données orientée document NoSQL qui est spécialement tournée vers une utilisation sur un système distribué. Elle garantit des transactions ACID sur des requêtes multi-documents ou sur tout un cluster, contient du clustering de manière native et peut tourner sur beaucoup de systèmes différents. Elle comporte des fonctionnalités avancées comme l'indexation dynamique, le full-text search ou bien le tri flexible.

Sa première version a été publiée en 2008 sous le nom de « Rhino DivanDB » et est codé grâce au langage C#. RavenDB propose également un service cloud (DBaaS) qui se nomme RavenDB Cloud. Elle supporte des clients dans plusieurs langages dont C#, C++, Java, NodeJS, Python, Ruby et Go.

Gestion de la réplication

La réplication dans RavenDB est implémentée à l'aide de clusters. Ceux-ci sont des regroupements de nœuds, un nœud étant une instance de RavenDB (typiquement sur des machines différentes). À l'intérieur de ces clusters, il peut y avoir plusieurs bases de données qui seront donc distribuées sur les différents nœuds. Pour garantir la cohérence au sein d'un cluster, RavenDB utilise leur protocole appelé Rachis, qui est dérivé du protocole de consensus pour système distribué nommé Raft.

Théorème de CAP

Nous avons vu en cours que le théorème de CAP disait que nous ne pouvions avoir que deux des propriétés suivantes dans un système :

- Consistent
- Available
- Partition tolerant

Et comme dans un système réel nous sommes forcément vulnérables au partitionnement, nous avons le choix entre être CP (consitent et partition tolerent) ou alors AP (available et partition tolerant). RavenDB offre la possibilité d'être CP et AP en même temps, cependant à des couches différentes. On distingue principalement deux types d'opérations dans RavenDB : les opérations au niveau du cluster (CP) et les opérations au niveau des bases de données (AP).

Pour les opérations sur les documents des bases de données, il est bien de pouvoir accéder en tout temps aux documents même si pas tous les nœuds du cluster sont disponibles et donc RavenDB a fait le choix d'appliquer la politique AP aux opérations CRUD sur les documents. Pour cela, il utilise la réplication multi-maîtres afin de permettre à chaque nœud d'accepter les écritures et donc de rendre le système hautement disponible mais éventuellement cohérent.

De l'autre côté, les opérations concernant l'entièreté du cluster doivent être consistantes afin de garantir la stabilité du système. Ces opérations peuvent être par exemple la création d'une nouvelle base de données ou encore la création d'un index. Afin de rendre ces opérations CP, RavenDB utilise leur propre protocole appelé Rachis et qui dérive du protocole déjà existant Raft.

Fonctionnement de Raft

Un nœud dans le protocole Raft peut avoir trois états : suiveur, candidat et leader. Au démarrage de l'algorithme, tous les nœuds sont des suiveurs. Quand un nœud est un suiveur, il démarre un timeout d'une durée aléatoire et qui sera réinitialisé à chaque fois qu'il reçoit un message du leader. Quand celui-ci arrive à terme (comme il n'a pas encore de leader), il va passer à l'état de candidat et envoyer une demande de vote à chaque autre nœud. Ceux-ci vont répondre et si le candidat reçoit le vote de la majorité des autres nœuds, il devient alors le leader. A partir de maintenant, les changements du système seront opérés par le leader.

À chaque fois que le leader reçoit une demande d'écriture, il va l'ajouter à ses logs et ensuite la transmettre aux suiveurs pour qu'ils ajoutent également la demande à leur propre log. Quand cela est fait, ils vont envoyer une réponse au leader pour l'informer qu'ils ont pu ajouter la demande à leur log. Quand le leader a reçu une réponse de la majorité des suiveurs, il va écrire la modification chez lui et ensuite notifier les suiveurs qu'il a pu commit la demande. Les suiveurs vont pouvoir donc à leur tour commit la demande et donc le système se trouvera dans un état cohérent. Ce processus s'appelle la réplication de log.

En plus du timeout des suiveurs, le leader a également son propre timeout (plus court que celui des suiveurs). Quand celui-ci se finit, il envoie un message (*append entries*) aux suiveurs et ceux-ci vont répondre à chacun de ces messages pour signaler qu'ils sont toujours présents. C'est le *heartbeat*. Les messages que le leader envoie aux suiveurs (ajout d'entrée dans les logs et commit) passent par ces messages du heartbeat.

Imaginons maintenant un cluster comportant cinq nœuds et que le leader et un suiveur (qu'on va nommer le groupe A) de ce cluster se trouve soudainement séparés des trois autres (groupe B). Comme le groupe B se retrouve sans leader (ils ne reçoivent plus les messages de heartbeat), ils vont élire un nouveau leader. En faisant une nouvelle élection, ils vont augmenter le *term*, c'est-à-dire le numéro de l'élection. Maintenant, si le leader du groupe A reçoit une demande d'un client, il recevra uniquement la réponse du suiveur qui est avec lui et donc la demande va rester uncommit dans ses logs car la majorité n'a pas été atteinte. En revanche si le nouveau leader du groupe B reçoit une demande, il va pouvoir la commit car plus de la moitié des nœuds (lui comprit) a pu écrire dans ses logs la demande. Quand la connexion se rétablit entre les deux groupes, le leader du groupe A va recevoir le message heartbeat du groupe B qui a été élu dans une élection plus récente que la sienne et redevenir un suiveur. Les modifications apportées pendant la séparation au groupe A vont être rollback et les nœuds de ce groupe vont être notifié des modifications des logs du groupe B. Le système se retrouve maintenant à nouveau dans un état global cohérent.

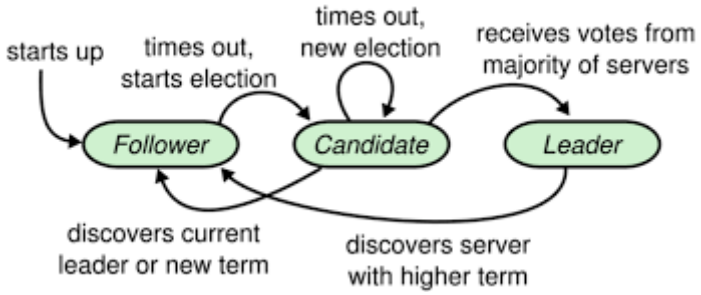


Figure 1 Diagramme d'état des rôles de RAFT

Rachis

Rachis est l'implémentation du protocole RAFT dans RavenDB. Il ajoute plusieurs fonctionnalités tel que :

- Le support de large machine d'état multitâche en mémoire et persistante
- Mise à jour fiable pour des groupes de machine d'état distribué
- Possibilité d'avoir des membres de cluster qui peuvent voter et d'autre pas
- Topologie dynamique, des nœuds peuvent être ajouter ou retiré à chaud
- Gestion de situation comme un leader injoignable ou bien forcé un leader à se retirer
- Utilisation du système de stockage Voron pour les logs locaux

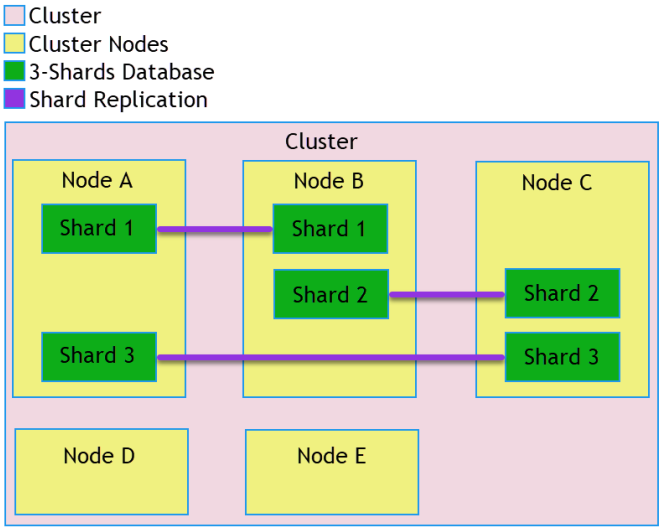
Système de sharding

RavenDB propose un système de sharding depuis la version 6.0 et chaque shard peut être répliquée sur différent nodes. L'utilisation du sharding est recommandée pour gérer une base de données RavenDB qui devient trop grosse à gérer pour un seul node au point ou les ressources comme la RAM, CPU ou le stockage ne peuvent plus suivre et que les fonctions basiques comme l'indexation ou les backups se transforment en des tâches lourdes à exécuter.

À noter que le sharding n'est totalement disponible que pour les licences *Entreprise* de RavenDB car les autres ne peuvent mettre les shards que sur un même node ce qui en réduit l'intérêt.

Une fois shardée l'utilisation de la base de données ne changent pas pour les clients, ils utiliseront toujours le même API que pour une utilisation non-shardée. De plus, même si le sharding n'est disponible que depuis la version 6.0, les utilisateurs utilisant un client de version antérieure n'auront aucun problème pour se connecter à la base de données et pour l'utiliser.

Le système repose sur le fait que des nodes de RavenDB vont jouer le rôle d'orchestrateur pour toutes les communications entre le client et les shards, c'est ce qui permet l'utilisation de l'API normalement utilisé sur les bases de données non-shardée. Toutefois, la présence de ces orchestrateurs implique un coût overhead plus conséquent car il y aura des communications en plus entre l'orchestrateur et le client avec cette implémentation.



Il est possible également de faire de la réplication pour les shards. Le nombre de réplication est géré par le *Shard replication factor* qui donne le nombre de répliques qui doit être maintenue pour chaque shards.

RavenDB conseille l'utilisation du sharding pour les bases de données qui ont un volume aux alentours de 250GB afin que la transition en base de données shardée soit bien mise en place quand ou si elle atteint 500GB.

Il est d'ailleurs assez simple de migrer une base de données non-shardée vers une qui l'est grâce à l'utilisation des fichiers *.ravendump* qui contient le contenu actuel de la base de données. Ce fichier peut ensuite être importer dans une nouvelle base de données shardée ou non.

Comme écrit plus haut les clients peuvent interagir avec les shards comme si elles n'étaient qu'une seule base de données mais les actions spécifiques sur les shards sont malgré tout disponible comme traquer une shard pour faire des requêtes directement dessus.

Gestion de base du sharding

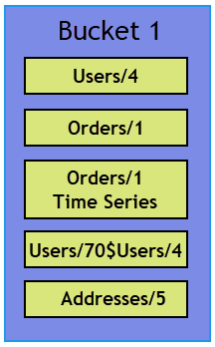
Les documents stockés dans une base de données shardées se trouvent dans des conteneurs virtuels appelé *Buckets*. Quand une base de données shardée est créée, le cluster alloue 1'048'576 (1024x1024) buckets pour la base de données entière qui seront ensuite répartit par groupe dans les différentes shards.

Pour remplir les buckets on applique un algorithme de hachage (comme xxhash) sur les ids des documents de manière à leur donner un id pour la répartition entre 0 et 1'048'575 afin de définir dans quel bucket le fichier va être sauvegardé et, comme les buckets sont déjà répartits entre les shards, cet id définit également à quelle shards le document appartient. Comme nous utilisons les ids des documents (uniques) la répartition entre les shards est homogène et ils auront également tous les documents qui leur sont liés sur la même shard (tel que les *Revisions,* les *Attachement,* etc.).

Anchoring documents

On peut décider d'ancrer un document à un buckets spécifique afin de regrouper des documents souvent requêtés ensemble dans la même transaction. Cela se fait simplement en ajoutant le symbole "$" suivit d'un suffixe que nous pouvons définir. RavenDB n'appliquera que la fonction

du hachage sur la partie qui suit le dernier ”$” quand il fait la répartition entre buckets, il suffit donc de donner le même suffixe à tous les documents ou mettre en suffixe le nom d’un document autour duquel on veut regrouper d’autres documents pour les avoir dans le même bucket.



Sharding by prefix

Le sharding de base ne nous donne pas beaucoup de contrôle sur la manière dont les documents sont répartis entre les shards, elle est efficace pour les répartir de manière homogène mais si l’on veut mettre des documents dans une shards qui est spécifiquement proche géographiquement d’un client, optimiser la performance des requêtes (éviter de requêter toutes les shards), etc. Il faut alors utiliser le sharding par préfixe, ce qui va changer la gestion de base du sharding.

Le préfixe sharding marche en définissant quel préfixe va dans quelle(s) shard(s) puis en rajoutant ce même préfixe aux id des documents que l’on veut sur ces shards spécifiques. Si un préfixe est associé à plusieurs shards, les documents seront alors répartis homogènement entre ces shards. On peut donner en tout 4096 préfixes différents, ils sont sensibles à la casse, doivent finir par “/” ou “-” et sont priorisé du plus spécifique au plus général (*user/eu/swiss/* définira la shards même si le préfixe *user/eu/* existe).

Il est important de savoir que RavenDB va donc réserver 1’048’576 buckets en plus pour chaque préfixe créé. Le document sera ensuite mis dans le numéro de bucket qui correspond au hachage de son document id comme il l’aurait fait sans préfixe mais dans les buckets réservés à son préfixe.

La grande différence avec l’ancrage de document est donc que nous pouvons choisir sur quelle shard nous voulons mettre les documents et pas que garantir qu’ils sont sur le même bucket.

Backup et restore

Du point de vue utilisateur le backup d’une base de données shardée se fait en lançant périodiquement une seule tâche de backup définie par l’utilisateur comme dans une non-shardée.

Dans une base de données non-shardée chaque cluster node à une réplication de la base de données et peut ensuite être responsable des tâches de backup pour backup la base de données entière.

Dans une base de données shardée chaque shards a la réplique d’une partie de la base de données, il n'y a donc pas un seul node qui peut gérer le backup de la base de données entière. Quand un utilisateur défini une tâche de backup RavenDB en crée automatiquement une par shards qui vont ensuite chacune définir un node qui va s’occuper de l’exécution de cette tâche. Chaque tâche de backup d’une shard peut choisir de garder le backup de sa partie de la base de données localement et/ou en ligne sur un service de cloud.

Bien qu’il existe deux types de backup dans RavenDB, seulement le *logical backup* est supporté pour les bases de données shardées (l’autre étant le snapshot).

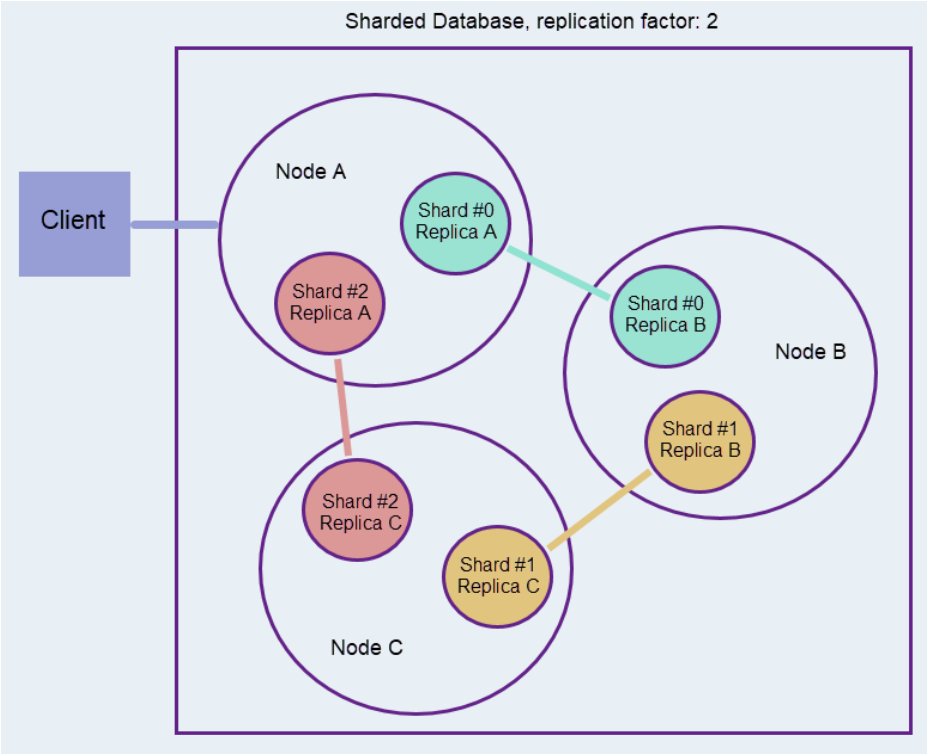
La restauration des données peut se faire sur la totalité de la base de données ou seulement sur une partie en trouvant les différences entre la base de données actuelle et celle backup.

Resharding

Ce système nous permet de bouger les données d'une shards à un autre, ce qui nous permet de maintenir un équilibre entre les shards dans le but de garantir que chaque shards s'occupe du même volume de donnée.

Le Resharding bouge toutes les données d'un bucket dans une nouvelle shard et associe ce bucket à cette nouvelle shard. Il ne peut, pour l'instant, qu'être lancé manuellement par l'utilisateur au travers du *Studio* RavenDB et quand il est lancé il se fait graduellement un bucket après l'autre de manière à limiter l'utilisation de ressources.

Exemple d'utilisation :



1. Le client demande de redistribuer les buckets de la shard #0 vers la shard #2.
2. La shard #0 se connecte à la shard #2 et lui transfère tout le contenu du premier bucket.
3. La shard #0 reste propriétaire du bucket jusqu'à ce que toutes les données aient été propagées à toutes les répliques de la shard #2.
4. La propriété est transférée et le bucket est remappé à la shard #2.
5. La shard #0 commence à purger toutes les entités dont la propriété est désormais détenue par la shard #2.
6. S'il reste des buckets à déplacer, la shard #0 peut commencer à transférer le contenu du bucket suivant.

Il se peut qu'un document se retrouve dans un bucket après que sa possession a été donnée et avant que celui –ci ne se fasse supprimer, dans ce cas une tâche routinière, la *periodic documents migrator,* vérifie le système et réinitialise un resharding si elle trouve une occurrence de ce type.

Fonctionnalités non supportées

Il y a malgré tous des fonctionnalités qui ne sont pas implémentées dans RavenDB pour les bases de données shardées alors qu'elles le sont pour celle qui ne sont pas shardées. Certaines fonctionnalités d'indexation ne sont pas supportées, tel que *Rolling index deployment,* faire des trieurs customisés ou charger un document depuis une autre shard. Il manque aussi des fonctionnalités sur les requêtes comme l'intersection, le *Highlighting,* l'ordonnancement par distance ou par score et plus encore. Il y a encore d'autre type de fonctionnalités qui manque dans d'autre thèmes comme la migration, l'importation et l'exportation, le patching, etc.

Conclusion

En résumé, RavenDB est une base de données orientée document qui offre une haute disponibilité pour la lecture / écriture des documents et une consistance éventuelle. Et ce qui concerne les opérations qui s'applique au niveau du cluster, RavenDB utilise le protocole de consensus Ratchis pour garantir la consistance. Le sharding est pris en cahrge par RavenDB et permet à l'utilisateur d'utiliser une base de données shardée comme si elle ne l'était pas et s'occupe automatiquement de répartir de façon homogène les documents entre ces shards, ce qui simplifie la vie des utilisateurs. Malgré cette automatisation, il y a quelques possibilités de contrôle comme celles présenté plus haut avec l'ancrage des documents ou le sharding par préfixe.

Webographie

https://blog.container-solutions.com/raft-explained-part-23-overview-core-protocol (22.11.2024)

http://thesecretlivesofdata.com/raft/#home (22.11.2024)

https://ravendb.net/docs/article-page/6.2/csharp/server/clustering/rachis/what-is-rachis (22.11.2024)

https://ravendb.net/learn/inside-ravendb-book/reader/4.0 (23.11.2024)

https://ravendb.net/docs/article-page/6.2/csharp/sharding/overview (23.11.2024)

https://www.youtube.com/watch?v=8xeRVCnxLVg (23.11.2024)