Universite Cote d'Azur/FDS(UEH)

MONGODB VS CASSANDRA

Etudiants :
Douilly RODELY
Djimy SURLIN
Pierre Rubens MILORME
Bob Charlemagne PIERRE

Professeur :
Gabriel MOPOLO-MOKE

September 26, 2024

Contents

1 Cassandra vs MongoDB : Une comparaison

Cassandra et MongoDB sont deux bases de données NoSQL populaires, mais elles ont des architectures, des cas d'utilisation et des caractéristiques de performance différentes. Voici une comparaison des deux :

1.1 Modèle de données

1.1.1 Cassandra

- Utilise un modèle `Wide-column store`, également appelé `familles de colonnes`.
- Les données sont stockées dans des `tables` où les lignes peuvent avoir un nombre différent de colonnes (chaque ligne peut être partielle).
- Un schéma est requis, mais il est flexible.

1.1.2 MongoDB

- C'est une base de données `orientée documents`.
- Les données sont stockées sous forme de `documents BSON (Binary JSON)`, qui sont flexibles et peuvent varier en structure au sein d'une même collection.
- Le schéma est plus dynamique et flexible que Cassandra.

1.2 Langage de requête

1.2.1 Cassandra

- Utilise le `CQL (Cassandra Query Language)`, qui ressemble au SQL mais est plus limité.
- Se concentre sur des requêtes mappées à des partitions spécifiques, optimisées pour un débit d'écriture élevé et des lectures prévisibles.
- Pas adapté pour des jointures complexes ou des requêtes multi-tables.

1.2.2 MongoDB

- Utilise un langage de requête riche et flexible basé sur `JSON`.
- Supporte des requêtes complexes, y compris des filtres, des agrégations, et l'indexation sur des champs imbriqués.
- Plus puissant pour des opérations multi-documents, avec des capacités telles que les jointures (via `$lookup`) et les pipelines d'agrégation.

1.3 Scalabilité et Architecture

1.3.1 Cassandra

- Très scalable avec une architecture sans maître.
- Décentralisé, ce qui signifie que tous les nœuds dans un cluster sont égaux et peuvent gérer à la fois les opérations de lecture et d'écriture.
- Les données sont réparties entre les nœuds à l'aide d'un hachage consistant et de la réplication.
- Scalabilité linéaire : Ajouter plus de nœuds augmente le débit de manière linéaire.

1.3.2 MongoDB

- Suit une architecture distribuée(sharding) pour la scalabilité horizontale.
- Peut supporter à la fois des ensembles de réplicas (pour la haute disponibilité) et le sharding (pour la scalabilité horizontale).
- MongoDB a un modèle primaire-secondaire, ce qui signifie que les écritures se font principalement sur le nœud primaire, tandis que les lectures peuvent être réparties sur les nœuds secondaires.

1.4 Performance

1.4.1 Cassandra

- Optimisé pour les charges de travail axées sur l'écriture.
- La vitesse des écritures et des lectures est prévisible et constante à grande échelle.
- Débit d'écriture élevé grâce à son moteur de stockage basé sur un journal en mode `append-only`.
- Idéal pour les données de séries temporelles, les journaux et les applications IoT où la performance d'écriture est cruciale.

1.4.2 MongoDB

-
- Performant pour les charges de travail axées sur la lecture, en particulier lorsque des requêtes complexes ou des agrégations sont nécessaires.
- L'indexation et les requêtes riches peuvent affecter les performances d'écriture par rapport à Cassandra.
- Convient aux applications nécessitant un schéma flexible et des requêtes basées sur des documents (par exemple, plateformes de commerce électronique, systèmes de gestion de contenu).

1.5 Cohérence et Disponibilité

1.6 Cassandra

- Priorise la disponibilité et la tolérance aux partitions plutôt que la cohérence forte (AP dans le théorème CAP).
- Offre une cohérence réglable : vous pouvez choisir des niveaux de cohérence par requête (par exemple, quorum, one, all).
- Adapté aux applications pouvant tolérer une cohérence éventuelle.

1.6.1 MongoDB

- Se concentre principalement sur la cohérence et la disponibilité (CA dans le théorème CAP).
- Les données sont fortement cohérentes pour les écritures sur le nœud primaire, avec des options pour les préférences de lecture sur les ensembles de réplicas.
- Peut offrir une cohérence éventuelle lors de la lecture à partir de nœuds secondaires.

1.7 Cas d'utilisation

1.7.1 Cassandra

- Bases de données de séries temporelles, plateformes de journalisation à grande échelle, applications IoT.
- Cas d'utilisation nécessitant un débit d'écriture élevé avec des modèles de lecture prévisibles.
- Applications nécessitant une tolérance aux pannes et une haute disponibilité dans plusieurs centres de données.

1.7.2 MongoDB

- Gestion de contenu, catalogues de produits, profils utilisateurs.
- Applications nécessitant des modèles de données flexibles, des requêtes `ad hoc` et une indexation riche.
- Cas d'utilisation où des requêtes complexes, des agrégations et un stockage de données de type `JSON` sont importants.

1.8 Réplication et Tolérance aux pannes

1.8.1 Cassandra

- Forte tolérance aux pannes grâce à son architecture décentralisée et sans maître.
- La réplication entre les nœuds (y compris la réplication multi-centres de données) est facile.

- En cas de panne d'un nœud, les autres nœuds prennent automatiquement en charge ses responsabilités.

1.8.2 MongoDB

- Tolérance aux pannes via des ensembles de réplicas.
- Basculement automatique vers des nœuds secondaires en cas de défaillance du nœud primaire.
- La réplication multi-centres de données est possible mais plus complexe à configurer que dans Cassandra.

1.9 Communauté et Écosystème

1.9.1 Cassandra

- Fort soutien de la communauté open-source et de la `Fondation Apache`.
- Des outils comme `DataStax` offrent un support commercial et des fonctionnalités supplémentaires.
- Moins d'intégrations préconstruites que MongoDB.

1.9.2 MongoDB

- Soutenu par `MongoDB Inc.` avec une grande et active communauté.
- Dispose d'un riche écosystème d'outils, de pilotes et d'intégrations officielles.
- Disponible à la fois en version open-source et en version cloud entièrement gérée (MongoDB Atlas).

1.10 Résumé des principales différences

Caractéristique	Cassandra	MongoDB
Modèle de données	Magasin en colonnes larges	Magasin de documents
Langage de requête	CQL	Langage de requête JSON riche
Scalabilité	Sans maître, scalabilité linéaire	Basée sur le sharding
Performance d'écriture	Élevée	Modérée
Performance de lecture	Cohérente, lectures simples	Puissante, requêtes complexes
Cas d'utilisation principaux	Axé sur l'écriture, séries temporelles, IoT	Schéma flexible, axé sur les documents
Modèle de cohérence	Cohérence éventuelle, réglable	Cohérence forte (primaire-secondaire)
Réplication	Décentralisée, tolérante aux pannes	Ensembles de réplicas, basculement manuel
Meilleur pour	Charges de travail intensives en écriture	Requêtes complexes, schémas flexibles

1.11 Conclusion

1.11.1 Cassandra

Cassandra est idéal pour les applications nécessitant un débit d'écriture élevé, une scalabilité et une disponibilité, avec des modèles de lecture plus simples. Il est parfait pour les données de séries temporelles ou les charges de travail IoT.

1.11.2 MongoDB

MongoDB excelle dans les scénarios où un schéma flexible, des requêtes complexes et une indexation riche sont nécessaires. Il est adapté aux applications comme les CMS, les plateformes de commerce électronique, et les cas d'utilisation avec des structures de données dynamiques.