Bases de Données NoSQL

Compte Rendu TP

Partitionnement MongoDB

Rayane Sendjakedine

Année 2025-2026

1 Introduction

Dans ce TP, on a mis en place un cluster MongoDB shardé pour comprendre comment MongoDB gère le partitionnement des données. Contrairement à la réplication qu'on a vu au TP précédent (qui duplique les données pour la haute disponibilité), le sharding divise les données entre plusieurs serveurs pour gérer de gros volumes.

L'architecture qu'on a déployé avec Docker contient :

— Un config server (qui stocke les métadonnées)
— Deux shards (qui stockent les données)
— Un routeur mongos (qui redirige les requêtes)

2 Mise en place du cluster

2.1 Configuration Docker

J'ai créé un fichier `docker-compose.yml` avec les 4 services nécessaires :

```
version: "3.8"

services:
  configsvr:
    image: mongo:7.0
    container_name: configsvr
    command: mongod --configsvr --replSet replicaconfig
             --port 27019
    volumes:
      - ./data/configdb:/data/db
    ports:
      - "27019:27019"

  shard1:
    image: mongo:7.0
    container_name: shard1
    command: mongod --shardsvr --replSet replicashard1
             --port 27018
    volumes:
      - ./data/shard1:/data/db
    ports:
      - "27018:27018"

  shard2:
    image: mongo:7.0
    container_name: shard2
    command: mongod --shardsvr --replSet replicashard2
             --port 27017
    volumes:
      - ./data/shard2:/data/db
    ports:
      - "27017:27017"

  mongos:
    image: mongo:7.0
    container_name: mongos
    depends_on:
      - configsvr
      - shard1
      - shard2
```

```
    command: mongos --configdb replicaconfig/configsvr:27019
             --port 27020
    ports:
      - "27020:27020"
```

2.2 Initialisation

Après avoir lancé les conteneurs avec `docker-compose up -d`, j'ai dû initialiser chaque replica set :

```
# Config server
docker exec -it configsvr mongosh --port 27019
rs.initiate({
  _id: "replicaconfig",
  configsvr: true,
  members: [{_id: 0, host: "configsvr:27019"}]
})

# Shard 1
docker exec -it shard1 mongosh --port 27018
rs.initiate({
  _id: "replicashard1",
  members: [{_id: 0, host: "shard1:27018"}]
})

# Shard 2
docker exec -it shard2 mongosh --port 27017
rs.initiate({
  _id: "replicashard2",
  members: [{_id: 0, host: "shard2:27017"}]
})
```

Ensuite j'ai ajouté les shards au cluster via mongos :

```
docker exec -it mongos mongosh --port 27020
sh.addShard("replicashard1/shard1:27018")
sh.addShard("replicashard2/shard2:27017")
sh.status()
```

2.3 Activation du sharding

Pour activer le sharding sur la collection films :

```
sh.enableSharding("mabasefilms")
sh.shardCollection("mabasefilms.films", {"titre": 1})
```

J'ai utilisé le champ `titre` comme clé de sharding. Après avoir lancé le script Python d'insertion, j'ai pu observer avec `sh.status()` que les chunks se répartissaient entre les deux shards au fur et à mesure.

3 Réponses aux questions

3.1 Concepts de base

1. Qu'est-ce que le sharding dans MongoDB et pourquoi est-il utilisé ?

Le sharding c'est la technique pour distribuer les données d'une collection sur plusieurs serveurs (les shards). Ça permet de gérer des volumes énormes de données qui ne rentreraient pas sur un seul serveur, et ça améliore les performances en parallélisant les opérations.

2. Différence entre sharding et réplication ?

La réplication duplique les mêmes données sur plusieurs serveurs pour la haute disponibilité (si un serveur tombe, les autres prennent le relais). Le sharding divise les données entre serveurs pour augmenter la capacité de stockage. En gros : réplication = copie complète, sharding = morceaux différents.

3. Composants d'une architecture shardée ?

— **Shards** : stockent les données (chaque shard a une partie du dataset)
— **Config servers** : stockent les métadonnées (qui dit quel shard contient quelles données)
— **Mongos** : routeur qui reçoit les requêtes des clients et les redirige vers les bons shards

4. Responsabilités des config servers ?

Les config servers gardent en mémoire toutes les infos du cluster : quels chunks sont sur quels shards, les limites de chaque chunk, la config des shards, etc. Si les config servers tombent, le cluster ne peut plus fonctionner car mongos ne sait plus où sont les données.

5. Rôle du mongos router ?

Le mongos est le point d'entrée pour les applications. Il analyse les requêtes, regarde dans les config servers où sont les données, et envoie les requêtes aux bons shards. Si une requête concerne plusieurs shards, il récupère les résultats de chacun et les combine avant de renvoyer au client.

3.2 Clés de sharding et chunks

6. Comment MongoDB décide sur quel shard stocker un document ?

MongoDB utilise la valeur de la clé de sharding du document pour déterminer à quel chunk il appartient, et donc sur quel shard il va. Par exemple avec ma clé "titre", tous les films dont le titre commence par A-M iront peut-être sur shard1, et N-Z sur shard2.

7. Qu'est-ce qu'une clé de sharding et pourquoi est-elle essentielle ?

C'est le champ (ou les champs) qu'on choisit pour partitionner les données. C'est crucial parce que ce choix détermine comment les données seront distribuées. Une mauvaise clé peut créer des déséquilibres où un shard reçoit beaucoup plus de données que les autres.

8. Critères d'une bonne clé de sharding ?

— Cardinalité élevée (beaucoup de valeurs différentes possibles)
— Distribution uniforme (pas de valeurs beaucoup plus fréquentes que d'autres)
— Non-monotone (pas toujours croissante comme un timestamp)
— Utilisée souvent dans les requêtes (pour pouvoir cibler un seul shard)

9. Qu'est-ce qu'un chunk ?

Un chunk c'est une plage de valeurs de la clé de sharding. Par exemple un chunk pourrait contenir tous les films de "A" à "D". Chaque chunk a une taille max (64 MB par défaut) et est stocké sur un shard spécifique.

10. Comment fonctionne le splitting ?

Quand un chunk atteint la taille max, MongoDB le divise automatiquement en deux chunks plus petits. Par exemple si le chunk A-D devient trop gros, il est splitté en A-B et C-D. C'est automatique et ça se passe en arrière-plan.

3.3 Balancer et performances

11. Que fait le balancer ?

Le balancer surveille la répartition des chunks entre les shards. Si un shard a beaucoup plus de chunks qu'un autre, le balancer déplace des chunks pour équilibrer la charge.

12. Quand et comment le balancer déplace des chunks ?

Il se déclenche quand la différence de nombre de chunks entre shards dépasse un seuil. Il copie alors les données d'un chunk du shard surchargé vers un shard moins chargé, puis met à jour les métadonnées dans les config servers.

13. Qu'est-ce qu'un hot shard et comment l'éviter ?

Un hot shard c'est un shard qui reçoit beaucoup plus de requêtes que les autres, ce qui crée un goulot d'étranglement. Pour l'éviter il faut choisir une clé de sharding qui distribue bien les accès, ou utiliser le hashed sharding.

14. Problèmes d'une clé monotone ?

Une clé qui augmente toujours (comme un timestamp ou un ID auto-incrémenté) fait que toutes les nouvelles insertions vont sur le même chunk, donc le même shard. Résultat : un shard fait tout le travail d'écriture pendant que les autres ne font rien.

3.4 Administration

15. Activer le sharding sur une base et une collection ?

```
sh.enableSharding("nom_base")
sh.shardCollection("nom_base.collection", {cle: 1})
```

16. Ajouter un shard ?

```
sh.addShard("replicaset/host:port")
```

Le balancer va ensuite redistribuer automatiquement des chunks vers ce nouveau shard.

17. Vérifier l'état du cluster ?

Les commandes principales :

— `sh.status()` : vue d'ensemble complète
— `db.stats()` : statistiques
— `sh.getBalancerState()` : état du balancer

3.5 Stratégies de sharding

18. Quand utiliser hashed sharding ?

Quand la clé de sharding est monotone (timestamp, _id). Le hash distribue uniformément les données même si les valeurs augmentent toujours. Par contre on perd la possibilité de faire des range queries efficaces.

19. Quand utiliser ranged sharding ?

Quand on fait souvent des requêtes sur des plages de valeurs. Par exemple pour des données géographiques ou temporelles. Les données proches sont sur le même shard donc la requête est plus rapide.

20. Qu'est-ce que le zone sharding ?

Ça permet de dire "ces données doivent aller sur ce shard spécifique". Utile pour la conformité RGPD (garder les données des utilisateurs EU en Europe) ou pour séparer des clients premium sur des serveurs plus performants.

3.6 Requêtes et optimisation

21. Gestion des requêtes multi-shards ?

Si la requête ne contient pas la clé de sharding, mongos doit faire un scatter-gather : il envoie la requête à tous les shards, attend les résultats de chacun, puis les combine. C'est beaucoup plus lent qu'une requête ciblée.

22. Optimiser les performances ?

— Toujours inclure la clé de sharding dans les requêtes si possible
— Créer des index appropriés

— Utiliser des projections pour limiter les données retournées
— Éviter les sorts et limits après l'agrégation multi-shards

23. Que se passe-t-il si un shard tombe ?

Si c'est juste un nœud dans le replica set, un autre nœud prend le relais. Si tout le shard tombe, les données de ce shard deviennent inaccessibles mais le reste du cluster continue. Les requêtes ciblant uniquement ce shard échouent.

24. Migrer une collection existante vers sharding ?

```
db.collection.createIndex({cle: 1})
sh.enableSharding("base")
sh.shardCollection("base.collection", {cle: 1})
```

MongoDB crée d'abord un seul chunk avec toutes les données, puis le split et distribue progressivement.

25. Outils de diagnostic ?

— `sh.status()` pour l'état global
— Les logs des conteneurs : `docker logs mongos`
— `db.currentOp()` pour voir les opérations en cours
— MongoDB Compass pour une vue graphique