



Université Sorbonne Paris Nord
Département d'informatique

Travaux Pratiques

TP n°3 : Le Partitionnement (Sharding) sous MongoDB

Denis Linde

17 décembre 2025

Table des matières

| | | |
|----------|---|----------|
| 1 | Introduction | 3 |
| 2 | Mise en place de l'architecture | 3 |
| 2.1 | Étape 1 : Préparation de l'environnement et Config Server | 3 |
| 2.2 | Étape 2 : Ajout des Shards au Cluster | 3 |
| 2.3 | Étape 3 : Configuration de la taille des Chunks | 4 |
| 2.4 | Étape 4 : Observation de la Migration (Balancer) | 4 |
| 3 | Réponses aux questions | 6 |

1 Introduction

L'objectif de ce TP est de mettre en œuvre un système de *sharding* (partitionnement) sous MongoDB. Nous allons déployer une architecture complète composée de trois éléments essentiels pour assurer la scalabilité horizontale :

- Un **Serveur de Configuration** (Config Server) : stocke les métadonnées du cluster.
- Un **Routeur** (Mongos) : dirige les requêtes vers les bons shards.
- Deux **Shards** (Serveurs de données) : stockent les fragments de données.

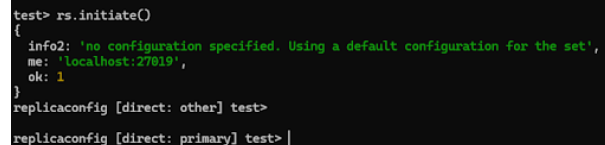
2 Mise en place de l'architecture

2.1 Étape 1 : Préparation de l'environnement et Config Server

Dans un premier temps, nous devons préparer l'arborescence de fichiers nécessaire au stockage des données de nos différents serveurs. Comme indiqué dans le sujet, nous créons trois répertoires distincts :

- `configsvrdb` : pour le serveur de configuration.
- `serv1` : pour le premier shard.
- `serv2` : pour le second shard.

Ensuite, nous démarrons le serveur de configuration. Ce serveur est crucial car il maintient la carte du cluster (quels chunks sont sur quels shards). Nous le lançons avec l'option `-configsvr` sur le port 27019.



```
test> rs.initiate()
{
  info2: 'no configuration specified. Using a default configuration for the set',
  me: 'localhost:27019',
  ok: 1
}
replicaconfig [direct: other] test>
replicaconfig [direct: primary] test> |
```

FIGURE 1 – Création des répertoires et lancement du Config Server

2.2 Étape 2 : Ajout des Shards au Cluster

Une fois les instances `mongod` des shards lancées (sur les ports 20004 et 20005) et le routeur `mongos` actif, nous devons déclarer ces shards au cluster. Cela se fait depuis le shell connecté au routeur (`mongos`) à l'aide de la commande `sh.addShard()`.

Nous ajoutons ici deux shards :

- `replicashard1` sur `localhost:20004`
- `replicashard2` sur `localhost:20005`

Le retour JSON `"shardAdded"` avec `"ok": 1` confirme que l'opération s'est déroulée avec succès.

```
[direct: mongos] test> sh.addShard("replicashard2/localhost:28085")
{
  shardAdded: 'replicashard2',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765982678, i: 5 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765982678, i: 5 })
}
[direct: mongos] test> sh.addShard("replicashard1/localhost:28084")
{
  shardAdded: 'replicashard1',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765982684, i: 5 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765982683, i: 39 })
}
```

FIGURE 2 – Ajout des deux shards via le routeur mongos

2.3 Étape 3 : Configuration de la taille des Chunks

Par défaut, la taille d'un *chunk* (fragment de données) dans MongoDB est de 64 Mo. Pour les besoins de ce TP, nous n'allons pas insérer des gigaoctets de données, mais nous voulons tout de même observer le mécanisme de *splitting* (découpage) et de migration des chunks entre les shards.

Pour faciliter cette observation avec un jeu de données réduit, nous modifions la configuration globale pour définir la taille des chunks à **1 Mo**.

Nous accédons à la base de données de configuration (`use config`) et nous sauvegarçons ce paramètre dans la collection `settings`.

```
[direct: mongos] test> use config
switched to db config
[direct: mongos] config> db.settings.updateOne(
...   { _id: "chunksize" },
...   { $set: { _id: "chunksize", value: 2 } },
...   { upsert: true }
... )
{
  acknowledged: true,
  insertedId: 'chunksize',
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
[direct: mongos] config> |
```

FIGURE 3 – Réduction de la taille des chunks à 1 Mo pour observer les migrations

2.4 Étape 4 : Observation de la Migration (Balancer)

Une fois le sharding activé, nous insérons des données (films) dans la collection. Initialement, MongoDB place les données sur un shard primaire. Cependant, dès que la taille des données insérées dépasse la limite de chunk fixée précédemment (1 Mo), le mécanisme de *splitting* découpe les données en nouveaux chunks.

Le **Balancer**, un processus d'arrière-plan du cluster, détecte alors un déséquilibre : un shard possède beaucoup de chunks alors que l'autre est vide. Il déclenche alors automatiquement la migration des chunks excédentaires vers le second shard.

Nous pouvons observer ce phénomène de deux manières :

- Via la commande `sh.status()` qui montre la distribution des chunks.

- En observant directement la taille des fichiers de données sur le disque : on voit que le dossier du deuxième shard commence à se remplir, prouvant que des données y sont transférées.

```
shardedDataDistribution
[
  {
    ns: 'mabasefilms.films',
    shards: [
      {
        shardName: 'replicashard2',
        numOrphanedDocs: 0,
        numOwnedDocuments: 8447,
        ownedSizeBytes: 1106557,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'replicashard1',
        numOrphanedDocs: 8411,
        numOwnedDocuments: 42365,
        ownedSizeBytes: 5465085,
        orphanedSizeBytes: 1085019
      }
    ]
  },
  {
    ns: 'config.system.sessions',
    shards: [
      {
        shardName: 'replicashard2',
        numOrphanedDocs: 0,
        numOwnedDocuments: 6,
        ownedSizeBytes: 594,
        orphanedSizeBytes: 0
      }
    ]
  }
]
```

FIGURE 4 – Observation du transfert de données vers le second shard

3 Réponses aux questions

1. **Sharding** : Le sharding est la méthode de partitionnement horizontal utilisée par MongoDB pour distribuer les données sur plusieurs machines afin de supporter de gros volumes de données.
2. **Différence avec la réplication** : La réplication duplique les données pour la haute disponibilité, tandis que le sharding divise les données pour augmenter la capacité de stockage et de traitement.
3. **Composants** : Une architecture shardée se compose de shards (stockage des données), de config servers (métadonnées) et de routeurs mongos (interface avec l'application).
4. **Config Servers** : Ils stockent les métadonnées du cluster et la cartographie des chunks, permettant au routeur de savoir où se trouvent les données.
5. **Mongos Router** : Il agit comme une interface de requête qui redirige de manière transparente les opérations des clients vers les shards appropriés.
6. **Décision de stockage** : MongoDB utilise la clé de sharding du document et les plages de valeurs définies dans les config servers pour déterminer le chunk de destination.
7. **Clé de sharding** : C'est un champ indexé choisi pour partitionner la collection ; elle est essentielle car elle détermine la répartition et l'équilibre des données.
8. **Critères de choix** : Une bonne clé doit avoir une cardinalité élevée, une distribution de fréquences uniforme et ne pas être monotone croissant.
9. **Chunk** : Un chunk est un bloc logique de données contiguës au sein d'une collection, défini par une plage de valeurs de la clé de sharding (min/max).
10. **Splitting** : Lorsqu'un chunk dépasse la taille configurée (ex : 64Mo), MongoDB le divise automatiquement en deux chunks plus petits sans déplacer les données physiquement dans l'immédiat.
11. **Balancer** : Le balancer est un processus d'arrière-plan qui redistribue les chunks entre les shards pour s'assurer qu'aucun serveur n'est surchargé par rapport aux autres.
12. **Déplacement des chunks** : Le balancer déplace les chunks lorsqu'il détecte une différence significative dans le nombre de chunks entre les shards, ou durant les fenêtres de maintenance configurées.
13. **Hot Shard** : C'est un shard qui reçoit une charge disproportionnée d'écritures ; on l'évite en choisissant une clé de hachage (Hashed Sharding) pour mieux répartir les insertions.
14. **Clé monotone** : Elle provoque l'écriture de toutes les nouvelles données sur un seul shard (le dernier), créant un goulot d'étranglement et annulant les gains de performance.
15. **Activation** : On active d'abord le sharding sur la base avec `sh.enableSharding()`, puis sur la collection avec `sh.shardCollection()` en précisant la clé.
16. **Ajout de shard** : On utilise la commande `sh.addShard("replicaset/host:port")` depuis le routeur mongos pour intégrer un nouveau serveur ou replica set.
17. **Vérification** : La commande `sh.status()` est la plus complète pour visualiser la topologie, la répartition des chunks et l'état du balancer.

18. **Hashed Sharding** : Il faut l'envisager lorsque la clé de sharding naturelle est monotone (comme un timestamp) afin de garantir une distribution uniforme des écritures.
19. **Ranged Sharding** : Il est à privilégier lorsque les requêtes portent souvent sur des plages de valeurs continues (ex : prix entre X et Y), afin de cibler un nombre réduit de shards.
20. **Zone Sharding** : Il permet d'associer des plages de clés spécifiques à des shards précis, ce qui est utile pour la localisation géographique des données (data locality).
21. **Requêtes multi-shards** : Si la requête ne contient pas la clé de sharding, le mongos l'envoie à tous les shards (scatter-gather) et agrège les résultats, ce qui est moins performant.
22. **Optimisation** : Pour optimiser les performances, il faut inclure la clé de sharding dans les filtres de requête afin que le mongos cible directement le bon shard.
23. **Indisponibilité** : Si un shard tombe en panne sans réplication fonctionnelle, la portion de données qu'il héberge devient inaccessible, provoquant des erreurs partielles.
24. **Migration existante** : On crée l'index sur la clé, on active le sharding, et le balancer redistribuera progressivement les données existantes entre les shards.
25. **Outils de diagnostic** : On utilise `sh.status()`, `mongostat`, `mongotop` ou MongoDB Atlas/Ops Manager pour surveiller la latence et la distribution des chunks.