Bases de données NoSQL

Nom: HAMRANI
Prénom: Amziane

I. Introduction:

La plupart d'entre nous avons déjà utilisé des bases de données relationnelles : ce sont celles où les données sont organisées en tables (lignes, colonnes), reliées entre elles par des clés primaires et étrangères, et interrogées avec le langage SQL. Ce modèle est très adapté quand les données sont bien structurées, que les règles d'intégrité sont fortes (contraintes, transactions ACID) et que l'on a besoin de requêtes complexes avec jointures.

Cependant, avec la montée en charge des applications web, des données massives et des besoins de très faible latence, ce modèle atteint ses limites en termes de performance et surtout de passage à l'échelle horizontale (ajouter facilement des machines). Pour répondre à ces contraintes, on a vu apparaître les bases de données dites *NoSQL*, qui regroupent plusieurs familles de SGBD non relationnels, pensés pour être plus flexibles sur le schéma et la cohérence afin de mieux optimiser la performance, la scalabilité et la gestion de données peu ou semi-structurées.

Les systèmes NoSQL se déclinent en plusieurs grandes familles, chacune adaptée à un type de données et de cas d'usage :

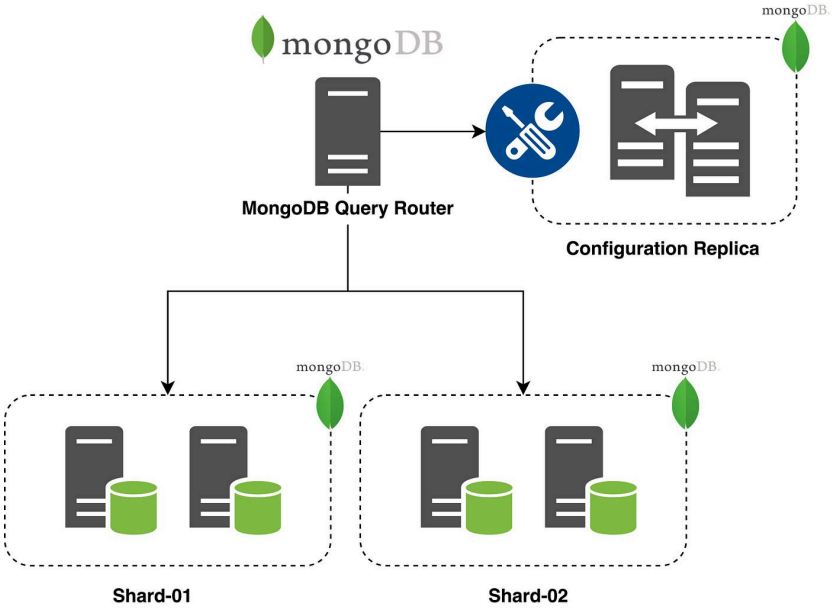
- Bases clé–valeur (comme Redis), qui s'appuie sur le stockage en mémoir, très simples et très rapides pour associer une clé à une valeur, souvent utiliser comme cache.
- Bases orientées documents (MongoDB), qui stockent des documents de type JSON.
- Bases orientées colonnes, utilisées pour de très grands volumes distribués.
- Bases orientées graphes, dédiées aux relations complexes entre entités.

Dans la suite du rapport, Redis sera présenté plus en détail comme un représentant de la famille clé–valeur.

II. Mongodb

MongoDB est une base de données NoSQL orientée documents, qui stocke les données sous forme de documents JSON (en réalité BSON, une version binaire) avec un schéma flexible. Cette flexibilité permet de facilement faire évoluer la structure des données sans devoir repenser intégralement la base. MongoDB est conçu pour être scalable horizontalement et performant, notamment sur des charges d'écritures importantes et des volumes de données très variés.

MongoDB utilise une architecture client-serveur distribuée où les données sont stockées sous forme de documents BSON dans des collections regroupées par bases de données. Le cœur est le processus mongod (serveur), secondé par mongos (routeur pour les clusters shardés) et les serveurs de configuration (métadonnées). Pour la scalabilité et la haute disponibilité, elle repose sur les Replica Sets (réplication primaire/secondaire avec basculement automatique) et le Sharding (partitionnement horizontal des données sur plusieurs nœuds).



Use Cases:

MongoDB est particulièrement adapté pour :

- Les applications nécessitant de gérer des données semi-structurées ou évolutives, comme les catalogues produits, les profils utilisateurs, ou les contenus web.
- Les solutions nécessitant une scalabilité horizontale importante, par exemple dans la télécommunication ou les systèmes IoT qui génèrent beaucoup de données.
- Les plateformes modernes d'IA ou d'exploitation de données non structurées grâce à la souplesse du modèle document, comme le montre la startup Ada avec MongoDB Atlas.

Installation:

Pour une installation rapide de MongoDB dans le cadre du TP, on utilise un conteneur Docker, ce qui évite de configurer le serveur nativement.

```
# 1. Récupérer l'image officielle MongoDB
docker pull mongo

# 2. Lancer un conteneur MongoDB en arrière-plan
docker run --name tp-mongodb -p 27017:27017 -d mongo

# 3. Se connecter au shell MongoDB
docker exec -it tp-mongodb mongosh
```

On peut également installer MongoDB via les binaires officiels pour chaque système d'exploitation (Linux, Windows, macOS). Rendez-vous sur la page d'installation officielle : https://www.mongodb.com/docs/manual/installation/

Manipulation:

Lecture de données : la commande find

En MongoDB, la lecture de données se fait avec find, qui joue le rôle de SELECT en SQL.

Syntaxe générale :

```
db.collection.find(<filtre>, <projection>)
```

- filtre = conditions (équivalent du WHERE)
- projection = champs à renvoyer (équivalent de la liste de colonnes)

Méthodes utiles sur le curseur retourné :

- `.sort({ champ: 1 | -1 })` : tri ascendant (1) ou descendant (-1)
- `.limit(n)` : limite le nombre de résultats
- `.skip(n)` : ignore les n premiers résultats
- `.pretty()` : affiche le résultat de façon lisible dans le shell.
-

Mise à jour de documents : updateOne et updateMany

En MongoDB, la mise à jour de données se fait avec updateOne et updateMany, qui joue le rôle de UPDATE en SQL.

db.collection.updateOne():

Modifier le premier document qui correspond au filtre.
Syntaxe générale :

```
db.collection.updateOne(filter, update, options)
```

- filter : document qui précise quel(s) document(s) cibler.
  Ex : `{ champ: valeur }`
- update : document qui décrit les modifications, avec des opérateurs :
  - `$set` : définir/modifier un champ
    `{ $set: { champ: nouvelleValeur } }`
  - `$inc` : incrémenter un champ numérique
    `{ $inc: { compteur: 1 } }`
  - `$unset` : supprimer un champ
    `{ $unset: { champASupprimer: "" } }`
- options (optionnel) : par ex. `{ upsert: true }` pour créer le document s'il n'existe pas.

db.collection.updateMany():

Modifier tous les documents qui correspondent au filtre.
Syntaxe :

```
db.collection.updateMany(filter, update, options)
```

- Mêmes paramètres que `updateOne`, mais appliqué à plusieurs documents.

Suppression de documents : deleteOne et deleteMany

db.collection.deleteOne()

Supprimer un seul document qui correspond au filtre.
Syntaxe :
`db.collection.deleteOne(filter)`

db.collection.deleteMany()

supprimer tous les documents qui correspondent au filtre.
Syntaxe :
`db.collection.deleteMany(filter)`

- Paramètre `filter` pour les deux : document de condition, comme pour `find`.

Agrégation : aggregate

db.collection.aggregate()

Effectuer des traitements avancés (groupement, statistiques, transformations).
Syntaxe :

```
db.collection.aggregate([ stage1, stage2, ... ])
```

- Paramètre :
  - tableau de “stages” (étapes) qui forment un pipeline.
- Stages courants :
  - `$match` : filtrer les documents (équivalent d’un WHERE).
    `{ $match: { champ: valeur } }`
  - `$group` : regrouper et calculer (COUNT, SUM, AVG, etc.).
    `{ $group: { _id: "$champGroupe", total: { $sum: 1 } } }`
  - `$project` : sélectionner / renommer / calculer des champs de sortie.
    `{ $project: { champ: 1, nouveauChamp: "$ancienChamp" } }`
  - `$sort` : trier les résultats.
    `{ $sort: { champ: 1 } }`
  - `$limit` : limiter le nombre de résultats.
  - `$unwind` : “déplier” un tableau pour avoir un document par élément.