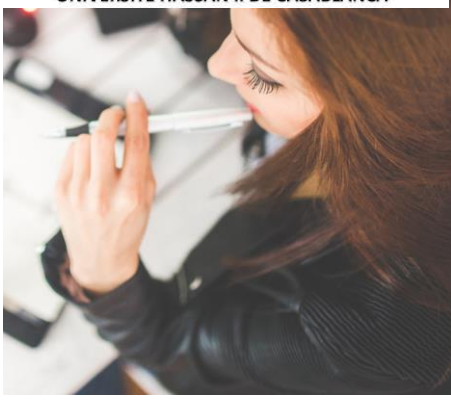
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

TD/TP 7 : LA SHARDING (OU LA REPARTITION DE DONNEES) SOUS MONGODB



Le but de ce TD/TP7 est la configuration d'un réseau de partitionnement de données capable de gérer la remonté en échelle.

Important : les manipulations sont consécutives, il est demandé d'utiliser plusieurs fenêtres console dont il faut les repérer à travers leurs titres.

:

1. CREATION DU SERVEUR DE CONFIGURATION

Pour créer le cluster, il vous faut tout d'abord un ConfigServer. Bien sûr, il est nécessaire de :

- Créer un répertoire dédié pour chaque serveur mongod (pas pour les mongos),
- Définir un port d'écoute à chacun. Les instructions suivantes sont à exécuter dans des **consoles différentes** :

```
mongod --configsvr --replSet configReplSet --port 27019 --dbpath C:\data\config1
mongod --configsvr --replSet configReplSet --port 27020 --dbpath C:\data\config2
```

Nous avons donc lancé deux serveurs de configuration (--configsvr) en ReplicaSet (configReplSet). Dans **une troisième console**, vous pouvez vous connecter à un de ces serveurs et initier le ReplicaSet :

```
mongo --port 27019
```

```
rs.initiate();
rs.add("local:27020");
```

2. CREATION DES SHARDS

Ensuite, il est nécessaire de lancer les shard en ReplicaSet. Nous allons en créer deux pour tester la distribution. Pour chaque shard, le paramètre --shardsvr est nécessaire pour permettre son intégration.

```
mongod --shardsvr --replSet sh1 --port 27031 --dbpath /data/sh1
mongod --shardsvr --replSet sh2 --port 27032 --dbpath /data/sh2
mongo --port 27031 --eval "rs.initiate()"
mongo --port 27032 --eval "rs.initiate()"
```

L'option --eval permet de faire passer la commande rs.initiate() directement au serveur puis de récupérer la main sur la console. Ici, cette commande permet d'initialiser le ReplicaSet de chaque shard.

3. LANCEMENT DU MONGOS ET CONNEXION DES SHARDS

Maintenant que nous avons des shards et des ConfigServers, nous pouvons nous attaquer au mongos (routeur).

```
mongos --configdb configReplSet/localhost:27019 --port 27017
```

Les shards peuvent alors être ajoutés les uns après les autres au niveau du mongos en mode console

```
mongo --port 27017
```

```
sh.addShard( "sh1/localhost:27031");
sh.addShard( "sh2/localhost:27032");
```

4. DISTRIBUTION DE LA BASE DE DONNEES

Ça y est, l'architecture de distribution est mise en place. Il suffit maintenant de définir la collection que l'on veut distribuer. Pour cela, nous allons créer une base « testDB » et une collection « test ».

```
use testDB;
sh.enableSharding("testDB");
db.createCollection("test");
db.test.createIndex({"_id":1});
sh.shardCollection("testDB.test",{"_id":1});
```

La collection test est maintenant distribuée automatiquement sur nos deux shards, les documents sont placés dans les chunks en fonction de leur valeur d'identifiant "_id", un tri de ces valeurs est effectué pour déterminer le placement (cf sharding arborescent). Nous pouvons maintenant importer les données des restaurants dans cette base de test :

```
mongoimport --db testDB --collection test --port 27017 restaurants_NY.json
mongo --port 27017 --eval "sh.status()"
```

Nous pouvons constater dans la partie « databases » que la collection testDB.test est composée de 4 chunks, répartis uniformément sur les deux shards.

5. STRATEGIE DE DISTRIBUTION

Nous avons vu comment créer une architecture de distribution et comment distribuer une collection. Maintenant, il serait intéressant de savoir comment choisir une stratégie de distribution appropriée à nos données, afin de fournir les meilleures performances pour nos requêtes.

Requêtes par intervalles

Jusqu'ici, nous nous sommes contentés de choisir l'identifiant « _id » pour distribuer les données. Par défaut, l'indexation est basée sur une index non-dense triant les données. De fait, les requêtes par intervalles de valeurs sont optimisées. Il est donc possible de choisir une autre clé pour les documents de la collection comme stratégie de distribution.

Pour profiter des requêtes par intervalles, nous pourrions créer une « clé de sharding » sur le code postal, la clé : zipcode. Ainsi, toute requête cherchant à retrouver les restaurants associés à ce code postal ou une plage de codes postaux seront efficaces.

```
sh.shardCollection("testDB.test2", {"address.zipcode" : 1});
```

Requêtes par zones

Le zipcode, c'est bien, mais comment mieux maîtriser la répartition des restaurants sur les différents chunks ? En effet, la technique précédente ne permet pas de contrôler la répartition physique des chunks et les intervalles de valeurs de ceux-ci. Le sharding par zone permet d'associer à chaque shard une zone. Tous les restaurants de cette zone seront alors alloués à un serveur dédié. L'avantage est double :

- Optimiser toutes requêtes liées à une zone (filtrage ou agrégation par quartier).
- Définir un serveurphysique dans un datacenterproche du client (zone géographique), correspondant à des requêtes de proximité.

Pour cela, il faut tout d'abord définir ces zones. Une zone est composée d'une plage de valeur et d'un « tag ». Nous pouvons choisir sur un intervalle de codes postaux correspondant aux villes :

```
sh.addTagRange("testDB.test2",
   {"address.zipcode": "10001"}, {"address.zipcode":  "11240"},
   "NYC")
sh.addTagRange("testDB.test2",
   {"address.zipcode": "94102"}, {"address.zipcode":  "94135"},
   "SFO")
```

L'avantage est de pouvoir associer des shards physiquement proches des utilisateurs dans chaque ville pour réduire la latence du réseau. Cela peut avoir également un intérêt si l'on souhaite stocker des données dans un cluster sécurisé.

Maintenant que les tags sont associés à des plages de valeurs, il suffit d'associer à chaque tag un shard. Un tag peut être associé à plusieurs shards.

```
sh.addShardTag("sh1", "NYC")
sh.addShardTag("sh2", "NYC")
sh.addShardTag("sh2", "SFO")
```

Automatiquement, les restaurants seront placés dans les chunks des shards désignés en fonction de leurs codes postaux. Il faudra toutefois rester vigilant à la répartition de charge et le dimensionnement des shards. Pour cela, vous pouvez consulter la répartition des chunks d'un tag dans le réseau :

```
use config;
db.shards.find({ "tags" : "NYC" });
```