

Bases de données documentaires et distribuées
Cours NFE04
Partitionnement par tri

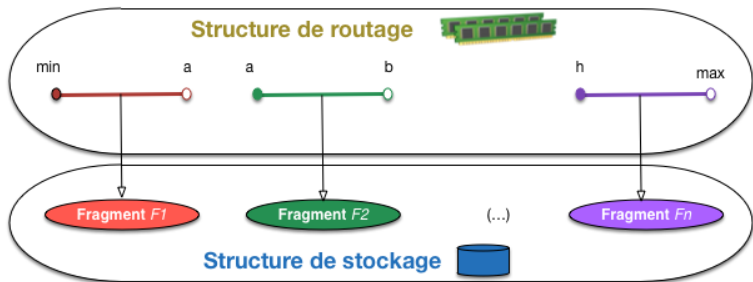
Auteurs : Raphaël Fournier-S'niehotta, Philippe Rigaux, Nicolas Travers
prénom.nom@cnam.fr

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Principe

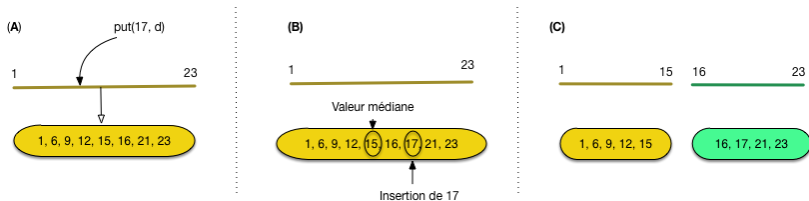
La collection est **triée** sur la clé et on la découpe en fragments d'une taille maximale pré-déterminée (de 64MO à qq GO)

Chaque fragment couvre donc un intervalle $[min_f, max_f[$.



Dynamicité : comment évolue le système ?

L'opération de base est le **split**, division d'un gros fragment en deux petits.



L'équilibrage consiste à répartir équitablement les fragments.

Efficacité du routage

La structure de routage est constituée de paires (I, a) où I est la description d'un intervalle et a l'adresse du fragment correspondant.

- si la taille d'un fragment est de 4 KO (choix typique d'un SGBD relationnel), le routage décrit 250 millions de fragments, soit une taille de 5 GO ;
- si la taille d'un fragment est de 1 MO, il faudra 1 million de fragments, et seulement 20 MO pour la structure de routage.

Dans tous les cas, le routage tient en mémoire RAM

Intervalle	Serveur
$] -\infty, a]$	A
$[a + 1, b]$	B
$[b + 1, c]$	C
$[c + 1, d]$	B
$[d + 1, \infty[$	A

Les opérations du routeur

get(k) : chercher l'intervalle qui contient k , transmettre au serveur correspondant.

put(k, d) : identifier le serveur avec k , transmettre l'insertion

delete(k) : idem

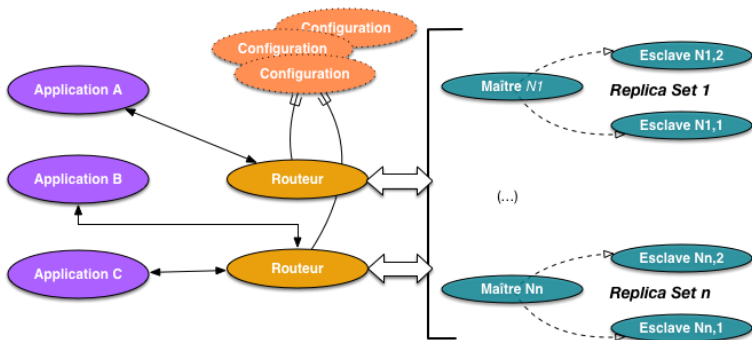
range(k_1, k_2) : transmettre à tous les serveurs gérant un intervalle chevauchant $[k_1, k_2]$

Toute autre opération : transmettre à tous les serveurs.

Intervalle	Serveur
$] -\infty, a]$	A
$[a + 1, b]$	B
$[b + 1, c]$	C
$[c + 1, d]$	B
$[d + 1, \infty[$	A

Le *sharding* de MongoDB

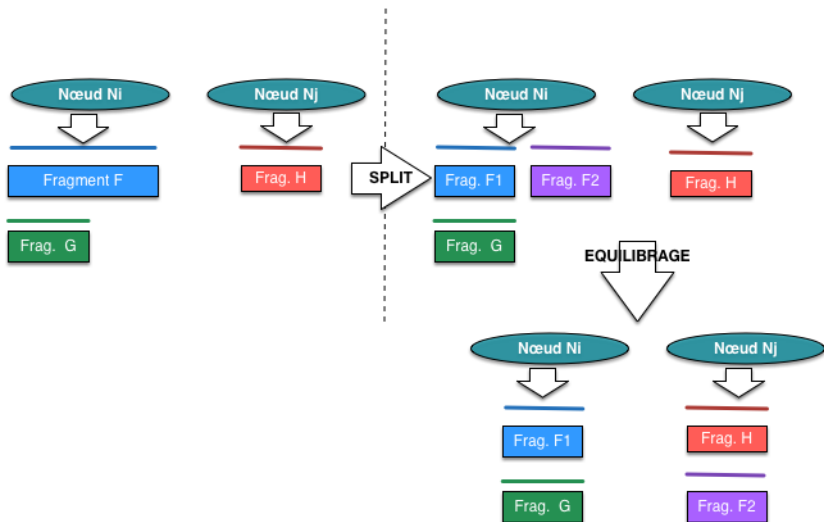
Le routeur connaît les intervalles, et les serveurs associés.



Remarque

- Chaque fragment est un *replica set* complet.
- On peut ajouter ou supprimer un serveur dynamiquement.

Partitionnement et équilibrage



Allons-y pour un petit test

On lance un *config server*

```
mkdir /data/configdb  
mongod --configsvr --dbpath /data/configdb --port 27019
```

On lance un routeur en lui indiquant où se trouve(nt) le(s) *config server(s)*

```
mongos --configdb localhost:27019
```

Et deux serveurs (il faudrait des *replica sets* complets en production)

```
mongod --dbpath /data/node1 --port=30001 --oplogSize 700  
mongod --dbpath /data/node2 --port=30002 --oplogSize 700
```

À partir de *mongos*, déclarons les deux serveurs comme *shards*.

```
mongo --port 27017  
sh.addShard("localhost:30000")  
sh.addShard("localhost:30001")  
sh.enableSharding("nfe204")
```


Choisir la clé de partitionnement

On indique la clé de partitionnement avec la commande suivante :

```
sh.shardCollection("nfe204.videos", { "title": 1 } )
```

Attention au choix de la clé

- Les clés-séquences : on envoie toujours les insertions au même serveur ; bon ou pas ?
- Les clés à faible cardinalité (ex : pays) : pas très bon non plus, car il pourra être difficile de faire un *split*

Bonne option : appliquer un hachage pour bien distribuer les insertions :

```
sh.shardCollection("nfe204.videos", { "title": "hashed" } )
```

Inspecter la configuration

Liste des *shards*

```
mongos> db.runCommand({listshards: 1})
```

Statut général du cluster

```
mongos> sh.status()
mongos> db.stats()
mongos> db.printShardingStatus()
```

Les fragments sont dans une collection *chunks*.

```
use config
db.chunks.count()
db.chunks.findOne()
```

Chargement en masse

Utiliser un générateur de données : ipsum
(<https://github.com/10gen-labs/ipsum>).

Charger 1 million de pseudo-films.

```
python ./pymonipsum.py -d nfe204  
    -c movies ---count 1000000 movies.jsch
```

Surveiller le chargement et les éclatements...