le cnam

Créer un cluster MongoDB

Travaux Pratiques

CNAM Paris
nicolas.travers (at) cnam.fr

Chapitre

Table des matières

Chapitre 1

Cluster MongoDB

1.1 Tolérance aux pannes

Nous voulons ici tester le système de tolérance aux pannes de MongoDB. Pour cela, nous allons lancer des serveurs avec des `ReplicaSets`.

1.1.1 Installation d'un replicaSet

Nous allons créer ici un replicaSet composé de 5 serveurs, dont 1 primaire, 3 secondaires et un arbitre.

1.1.1 À chaque serveur, un répertoire de données doit être créé :

- /data/rs1
- /data/rs2
- /data/rs3
- /data/arb

1.1.2 Choisir un nom de réplication : `RS1`

1.1.3 Le premier serveur sera lancé sur le serveur créé dans le précédent TP :
`mongod --replSet RS1 --dbpath /data/db --port 27017`

1.1.4 Ouvrir 4 nouvelles consoles

1.1.5 Dans chaque console[1], lancer un replicat avec un nouveau port de connexion :

- `mongod --replSet RS1 --dbpath /data/rs1 --port 27018`
- `mongod --replSet RS1 --dbpath /data/rs2 --port 27019`
- `mongod --replSet RS1 --dbpath /data/rs3 --port 27020`

1.1.6 Ouvrir une nouvelle console pour se connecter au serveur qui sera choisi comme serveur primaire :
`mongo --port 27017`

1.1.7 Lancer le mode de réplication :
`rs.initiate();`
Dans la console "mongo" apparaîtra `RS1:PRIMARY>` désignant le serveur primaire.

1.1.8 Rajouter chaque réplicat (en associant le "host" de votre machine) :

- `rs.add(``<host>:27018'');`
- `rs.add(``<host>:27019'');`
- `rs.add(``<host>:27020'');`

1.1.9 Vérifier la configuration de l'ensemble des serveurs :
`rs.conf();`

1.1.10 Lorsqu'il y a plusieurs réplicats, il est nécessaire de définir un arbitre dans le cas où le serveur primaire tombe. Ouvrir une nouvelle console :
`mongod --replSet RS1 --dbpath /data/arb --port 30000`

1.1.11 Ajouter l'arbitre dans la console mongo `RS1:PRIMARY>`
`rs.addArb(``<host>:30000'');`

1.1.2 Test de réplication

1.1.1 Dans la console `RS1:PRIMARY>` vérifier le contenu de la BD "DBLP" : `use DBLP;db.publis.count();`

1.1.2 Ouvrir une nouvelle console sur un serveur réplicat : `mongo --port 27018`
Dans la console mongo apparaîtra `RS1:SECONDARY>` désignant le serveur secondaire (réplicat).

1. Attention, au vu du nombre de consoles ouvertes, il sera bon de se rappeler chaque processus lancé (voir de renommer les terminaux)

1.1.3 La commande : `use DBLP;db.publis.count();` retourne une erreur car l'utilisation d'un réplicat n'a pas été autorisé. Pour ce faire, taper la commande : `rs.slaveOk();`
Refaire le test

1.1.4 Dans la console `RS1:PRIMARY>`, ajouter un document "`{ test : 1}`"

1.1.5 Dans la console `RS1:SECONDARY>`, vérifier son existance : `use DBLP;db.publis.find({test:1});`
Cela peut prendre quelques secondes (souvent quelques milli-secondes)

1.1.3 Test de tolérance aux pannes

1.1.1 Dans la console `RS1:PRIMARY>`, ajouter un document "`{ test : 2}`" à la collection DBLP.publis

1.1.2 Tuer le processus mongod "primaire" (console ouvrant mongod avec le port 27017) avec "ctrl+C"
Ou dans la console `RS1:PRIMARY>`, taper : `use admin;` puis `db.shutdownServer();`

1.1.3 Regarder le comportement des autres mongod, en particulier l'arbitre (port 30000). Sera alors désigné le serveur qui deviendra le primaire.
Il est aussi possible de connaître le serveur primaire dans la console `RS1:SECONDARY>` : `rs.status();`

1.1.4 Dans la console `RS1:PRIMARY>`, taper 'exit'. Relancer "mongo" mais avec le port du nouveau serveur primaire

1.1.5 Vérifier l'existence de votre document dans la collection DBLP.publis : `use DBLP;db.publis.find({test:2});`

1.1.4 Fichier de configuration

Nous souhaitons automatiser le lancement d'un serveur réplicat. Pour cela, nous allons créer un fichier de configuration (/data/mongo1.conf) Il est nécessaire de créer un fichier de configuration par serveur (chaque serveur en théorie est sur une machine distincte).

Voici à quoi ressemble un fichier de configuration :

```
# mongo1.conf
#Fichier de log du serveur. Le répertoire doit être créé et mongo doit avoir les droits d'écriture.
#Un fichier de log par serveur
logpath=/var/log/mongodb/mongod1.log
logappend=true

# Permet de reprendre la main dans le terminal lorsque le serveur est lancé
fork=true

# Paramètres de connexion et de stockage à changer pour chaque serveur
bind_ip=127.0.0.1
port=27017
dbpath=/data/db

# Fichier créé pour vérifier si le serveur tourne
pidfilepath=/var/run/mongodb/mongod.pid

# Niveau de "oplog" pour la réplication
#   0=off (default), 1=W, 2=R, 3=both, 7=W+some reads
diaglog=1

# Replication : Mettre le nom du replicaSet
replSet=RS1
# Taille maximum de l"oplog" pour la réplication
oplogSize=1024
```

L'ensemble des options du fichier de configuration se trouve ici :
`http://docs.mongodb.org/manual/reference/configuration-options/`

1.1.1 Créer un fichier de configuration pour chaque serveur du TP précédent

1.1.2 Lancer chaque serveur en utilisant le fichier de configuration correspondant :
`mongod --config /data/mongo1.conf`

1.1.3 Dans le cas où le replicaSet ne serait pas initialisé (ce qui n'est pas le cas actuellement), il est possible d'initialiser avec l'ensemble des serveurs :

```
rsconf = {
          _id: "RS1",
          members: [
                    {_id: 0,host: "<hostname>:27017"},
                    {_id: 1,host: "<hostname>:27018"},
                    {_id: 2,host: "<hostname>:27019"},
```

```
                    {_id: 3,host: "<hostname>:27020"},
                    {_id: 4,host: "<hostname>:30000",arbiterOnly:true},
                ]
        };
rs.initiate(rsconf);
```

1.2 Partitionnement des données

Nous souhaitons maintenant distribuer les données sur plusieurs serveurs de données.

1.2.1 Installation du sharding

1.2.1 Création du serveur de configuration

- Dans une console, créer le répertoire de configuration :
  `mkdir /data/configdb`
- Lancement du serveur (possibilité de faire un fichier de configuration)
  `mongod –configsvr --dbpath /data/configdb --port 28000`
- Création d'un routeur de *sharding* "mongos" lié au serveur de configuration :
  `mongos --configdb localhost:28000 --port 29000`

1.2.2 Lancement de trois serveurs de données (après création des répertoires) :

- `mongod --dbpath /data/sh1 --port 27017`
- `mongod --dbpath /data/sh2 --port 27018`
- `mongod --dbpath /data/sh3 --port 27019`

1.2.3 Connexion au routeur et installation du sharding pour deux serveurs :

- `mongo --port 29000`
- `sh.addShard(``localhost:27017'');`
- `sh.addShard(``localhost:27018'');`
- Créer la collection "publis" sur le routeur :
  ```
  use DBLP;
  db.createCollection(``publis'');
  db.createIndex({year:1});
  ```
- Permettre le sharding sur un BD : `sh.enableSharding(``DBLP'');`
- Donner la clé de distribution : `sh.shardCollection(``DBLP.publis'', year : 1);`
  Une fois choisi, nous ne pourrons changer le sharding (sinon en supprimant l'ensemble)
- Lancer le script d'importation en changeant le port du serveur vers **29000**. Ne pas se tromper surtout (sinon, pas de distribution)
- Votre cluster est maintenant créé.

1.2.2 Inspecter la configuration

- Consulter l'état de la distribution des données sur la console "mongos" (port 29000) :
  `sh.status();`
- Compter le nombre du documents dans la base :
  `db.publis.count();`
- Puis lancer une console mongo sur chaque serveur (27017, 27018) et compter :
  ```
  use DBLP
  db.publis.count();
  ```
  Vous constaterez une différence, pourquoi ?
- Sur la console "mongos" compter les publis de "Toru Ishida"
- Idem sur chaque console de serveur
- Ajouter un nouveau shard (port 27019). Constater les changements sur le status et la répartition sur chaque sur chaque serveur (cela peut prendre du temps)

1.2.3 Tolérance aux pannes & Sharding

— Tuer le processus d'un shard (27018)

— Compter le nombre de document sur la console "mongos".
Le processus est impossible a être effectué

— Créer trois *replicaSet* (RS1, RS2, RS3) vu comme dans la partie précédente (3 serveurs chacun, répertoire sh1r1, sh1r2, sh1r3, sh2r1…)

— Créer votre cluster (sharding) avec pour chaque shard le serveur primaire préfixé du replicaSet : `sh.addShard(``RS1/localhos`

— Vous pouvez maintenant tuer un processus, le replicaSet prend le relai