TP sur MongoDB

Inspiré des TPs de P.Rigaux et al.

Documents utiles:

- tutoriel docker: https://docs.docker.com/get-started/
- commandes docker: https://docs.docker.com/engine/reference/commandline/cli/
- toutes les informations utiles sur MongoDB se trouvent dans la doc, voire même souvent dans le guide pour débuter, mais des pointeurs plus précis vous sont fournis pour gagner du temps à chaque fois pour chaque question mettant en oeuvre un outil non présenté en cours.

Code couleur:
`docker help` : instruction à écrire dans le terminal
`help` : instruction à écrire dans le shell mongo.

Aide pour Docker (réseau)

`docker network create <nomreseau>` crée un réseau

`docker run --net <nomreseau> -p 27017:27017 --name <mc> <monimage>... --port 27017` publie sur le réseau les ports (syntaxe plus générale: ip:hostPort:containerPort).

`docker attach nomconteneur` attache l'entrée/sortie/erreur standard au conteneur.

Paramètres des serveurs mongo(d/s)

`mongo` le shell mongo, que l'on peut exécuter sur les serveur `mongod` et `mongos`

`mongod --replSet <nomset> --port <numport>` lance le serveur (daemon) `mongod`:

`--replSet` est nécessaire pour pouvoir connecter les conteneurs dans le replica set spécifié.
`--port` spécifie le port sur lequel le serveur est à l'écoute

En cas de sharding:
`--configserv` est nécessaire pour un serveur de configuration
`--shardsvr` est nécessaire pour un serveur stockant un fragment

`mongos --port 27017 --configdb <nomsetcfg>/<hote>:<numportcfg>` lance un routeur

Instructions du mongo shell

```
// Initialiser le replica set: depuis un des membres, on ajoute les autres serveurs
rs.initiate()
rs.add ("<hostname1>:<port1>")
rs.add ("<hostname2>:<port2>")

// permettre les lectures sur un serveur esclave (à taper sur l'esclave):
rs.slaveOk()

// ajouter des fragments (à taper sur le routeur):
sh.addShard("<nomset>/<hostname>:<port>")
```

1 Préparation de la base de donnée

1. Vérifier que docker est bien installé en lançant l'image `hello-world` (inutile de nommer le conteneur).
2. Une fois ceci fait, lister les images docker et conteneurs docker présents sur la machine.
3. Récupérer l'image mongodb: `mongo:latest`

   Pour plus d'informations sur l'image vous pouvez jeter un coup d'oeil à https://hub.docker.com/_/mongo/ mais ce n'est pas utile.
4. Lancer le serveur `mongod` en mode "détaché" dans un conteneur docker que vous appellerez `mongoserv`.[1]

[1] Vous laisserez docker choisir automatiquement l'emplacement des fichiers où seront enregistrées les données. Cela peut ralentir les opérations d'écritures mais évite d'avoir à configurer les droits d'accès pour les conteneurs dockers.

5. Lancer l'interpréteur de commandes (shell) `mongo`, c'est à dire le client, dans ce conteneur docker.

   ```
   docker exec -it mongoserv mongo
   ```

   > Le shell mongo est un interpréteur javascript. On peut donc lui passer aussi bien des instructions javascript que des commandes de MongoDB. Il accepte de nombreux raccourcis unix (Tab, Up, Ctrl+a, Ctrl+r, Ctrl+k, Ctrl+c...).

6. Demander à utiliser la base de donnée `bdtest`. Que se passe-t-il? La base est-elle créée?

7. Insérer un document quelconque dans la collection `movies`. Que se passe-t-il?

8. Il ne serait pas pratique d'insérer manuellement de grandes données. Nous allons donc utiliser l'utilitaire `mongoimport` permettant d'importer les données depuis des fichiers json ou csv.

   Télécharger le fichier contenant les mails d'Enron puis copier le fichier dans le conteneur mongoserv. Puis chargez le fichier dans la collection `emails` de la base de donnée `bdenron`.

   Vous utiliserez pour cela l'application `mongoimport` du conteneur mongoserv.

   > `mongoimport` fait partie des applications installées par l'image docker que nous avons récupéré. Cette application suppose par défaut que le fichier d'entrée contient un document JSON par ligne, sans virgule en fin de ligne. C'est le cas pour notre document. La syntaxe que nous utiliserons sera donc:
   >
   > `mongoimport --db <base> --collection <collection> --file <fichier.json>`
   >
   > Si le fichier à charger avait été un JSON array, on aurait ajouté l'option `--jsonArray`

2 Requêtes sous MongoDB

1. Connectez-vous à la base de donnée. Compter le nombre de documents dans la collection `emails`.

2. Qu'affiche `db.emails.find().pretty()` ? Effectuer quelques itérations du curseur.

3. Lister les courriels affichés par l'expéditeur (`sender`) `no.address@enron.com`.

4. Afficher le 10ème courriel par ordre alphabétique d'expéditeur, de deux manières:

   (a) en stockant le curseur renvoyé par `find()` dans une variable javascript, puis en parcourant ce curseur (par ex, en le transformant en vecteur, ou en itérant)

   (b) en utilisant les instructions `skip()` et `limit()` de MongoDB.

5. Lister les différents dossiers (`folder`) à partir desquels les courriels ont été extraits. Observer le type du résultat.

6. Afficher le nom de l'auteur et du destinataire de chaque courriel, en triant le résultat par auteur.

7. Afficher les messages reçus en 2000 après le mois d'Avril

   Indication: la date est stockée comme une chaîne de caractère.

8. Afficher les messages pour lesquels le champ "replyto" n'est pas nul.

   Indication: on pourra utiliser les opérateurs de requêtes `$not` *et* `$type`*, ou directement comparer à* ***`null`***

9. Nombre de courriels envoyés par chaque opérateur sur la période ci-dessus. Afficher le résultat en triant par nombre de courriels décroissant.

   Indication: le tri sera une des étapes du pipeline d'agrégation. En effet, il on ne peut pas effectuer l'opération de tri ***`sort()`*** *sur le curseur (un peu particulier) renvoyé par* `aggregate()`*. Si vous êtes curieux, voir ici.*

10. Recalculer la requête ci-dessus, mais en utilisant mapReduce (et sans le tri).

    Indication: commencer par calculer sans restriction sur la période. Pour filtrer la période, vous rajouterez une condition en utilisant la clause `query` *voir la documentation sur mapReduce*

11. En vous aidant de la doc et en expérimentant, vérifier ce que calcule la requête

    ```
    db.emails.find({   text: {$regex: '^(?si).*P(?-si)resident Bush.*$'}   })
    ```

3 MongoDB sur plusieurs noeuds: réplication

1. Lancer trois serveurs `mongod` sur les ports 27018, 27019, 27020 respectivement. Chacun sera bien entendu sur son propre conteneur. Ne les détachez pas, ou bien si vous les avez détachés, lancer `docker attach <nomconteneur>` pour pouvoir observer ce qui se passe dans le terminal.

2. Lancer l'interpréteur de commande `mongo` dans un des conteneurs, en précisant le port correspondant. Par ex: `docker exec -it mongoserv1 mongo --port 27018`

3. Initialiser le replica set en ajoutant les deux autres serveurs. L'adresse ip des trois est la même, et vous pouvez, par exemple, la voir affichée dans le résultat de `rs.initiate()`.

4. Vous pouvez consulter les participants au replica set par `rs.conf()`. Pour plus d'informations sur la synchronisation (vérifier le noeud primaire et les secondaires), utilisez `rs.status()` ou son équivalent `db.adminCommand( replSetGetStatus : 1 )`.

   - Observer l'intervalle entre deux heartbeat.
   - (sauter cette question si vous ne connaissez pas javascript)
     afficher uniquement les informations concernant les différents membres à partir de status, en utilisant une des notations javascript (dot ou bracket) pour accéder aux propriétés d'un objet javascript.
     Puis afficher uniquement pour chaque membre: son numéro, status, et nom au format:
     `"0: PRIMARY - a-140395.ups.u-psud.fr:27018"`

5. Lancer un interpréteur de commande mongo sur les deux autres esclaves (sur les conteneurs). Insérer les données enron. Vérifier que les données sont propagées depuis le serveur primaire (essayez directement de lire sur les autres serveurs, puis réessayez après `rs.slaveOk()`). Vérifier ce qu'il se passe si vous essayez d'écrire sur un esclave.

6. Arrêter le serveur primaire par la commande `db.shutdownServer()`) et observer la console ainsi que les nouvelles propriétés du replica set.

4 MongoDB sur plusieurs noeuds: partitionnement

1. Lancer un serveur de config (`mongod --configserv ...`). Votre serveur consistera en un replica set contenant un seul noeud primaire[2]. Puis initialiser le replica set.

   Je suggère d'utiliser les valeurs suivantes[3] pour les paramètres: écouter sur le port 27018, nom du replica set: serveurconfig, nom serveur: mconfig.

2. Lancer un routeur (`mongos --configdb ...`).

   Paramètres suggérés: écouter sur le port 27017, nom routeur: mrouteur.

3. Lancer deux serveurs de données pour contenir les données partitionnées (`mongod --shardsvr`). Chaque serveur de donnée sera encore une fois un replica set constitué d'un seul noeud. Initialiser les replica sets.

   Paramètres suggérés: écouter sur le port 27030 (respectivement 27031), nom des replica sets: rset1, rset2, nom serveurs: mshard1, mshard2.

4. Ajouter les deux fragments à la grappe gérée par le routeur en exécutant sur le routeur:

   `sh.addShard(<nomset> --port <numport>)`

   Si vous avez suivi les suggestions, à part le nom de machine qui n'est peut être pas a-140395:
   `docker exec -it mrouteur mongo --port 27017`
   puis
   `sh.addShard("rset1/a-140395.ups.u-psud.fr:27030")`
   `sh.addShard("rset2/a-140395.ups.u-psud.fr:27031")`

5. Observer l'état de vos bases de données avec

   `sh.status()`

   `db.printShardingStatus()`

   `db.stats()`

[2] ce n'est pas robuste de n'avoir qu'un noeud dans le replica set, donc à éviter en production mais pour simplifier ce tp on ne se préocuppe pas de la résistance aux pannes ici.
[3] mais je n'impose pas ces valeurs: vous pouvez en changer, mais ce sont les paramètres que j'utilise dans le corrig

6. créer une base de donnée `bdpart`, contenant une collection `dblp`. Autoriser le partitionnement des collections dans la base `bdpart`:

   ```
   db.adminCommand( { enableSharding:  "<database name>" } )
   ```

   Partitionner par hachage la collection `dblp` en fonction du champ "ident"

   ```
   sh.shardCollection("<base>.<collection>", {"clé":  "hashed"})
   ```

7. Récupérer et décompresser les données

   ```
   wget http://b3d.bdpedia.fr/files/dblp.json.zip; unzip dblp.json.zip
   ```

   Vous observerez que le fichier contient déjà un champ `_id` alors que ses valeurs ne conviennent pas pour un identifiant mongodb. Renommez donc ce champ en `ident`, par exemple avec `sed`.

8. Copier le fichier dans le conteneur du routeur, puis charger les données avec `mongoimport` en utilisant l'option `--jsonArray` vu la forme du document. Observer.

9. Écrire une requête qui va sélectionner par ordre alphabétique d'auteurs tous les documents de la collection `dblp` pour lesquels `"type" = "Article"`. Inutile d'exécuter cette requête mais analysez son plan d'exécution avec l'instruction `explain()` s'utilisant comme suit:

   ```
   <requête>.explain()
   ```

Avant de quitter la salle, exécuter impérativement le script suivant:

```
#!/bin/bash

# remove  containers:
docker ps -aq | xargs -r docker rm -f

# remove unused images (all, here, since all containers were killed):
docker images -{}-no-trunc | awk '{ print $3 }' | xargs -r docker rmi

# remove unused volumes:
docker volume ls -qf dangling=true | xargs -r docker volume rm
```

Et effacer les fichiers JSON que vous avez téléchargé.