NoSQL - MongoDB

Objectif

Découvrir l'agrégation des requêtes, l'indexation

Révisions

Téléchargez depuis le site le fichier `ex.tgz`. Décompressez dans un répertoire ex.

1. Importez le contenu du répertoire dans la base de données `esipe2`. Pour cela, utilisez la commande mongorestore (faire un –help pour voir les commandes).
2. Combien avez-vous de collections dans la base ?
3. Combien avez-vous d'enregistrements dans la collections ex ?
4. Afficher le contenu du document dont l'id (_id) est 99.
5. Afficher uniquement les items du document 99.
6. Afficher uniquement les quantités des items du document 99.
7. Combien y-a-t-il de documents dont le score est supérieur ou égale à 100

Agrégation

Dans un premier temps, on utilise le framework d'agrégation de Mongodb. En utilisant la page **http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/,** rédigez les requêtes suivantes :

8. La commande `distinct` permet d'obtenir les données distinctes d'une clé donnée. La syntaxe est la suivante : `db.runCommand({''distinct '' : nomCollection, ''key'' : cléCollection})` Ecrire la requête permettant d'obtenir l'ensemble des scores de la collection ex.
9. Calculez la somme des scores pour l'ensemble des documents
10. Calculez la somme des scores pour les documents dont les id sont inférieurs à 500.
11. En agrégeant par score, calculez la somme des scores pour les documents et afficher uniquement les totaux supérieurs à 4000.
12. Fournir la somme des quantités pour les élements du tableau items.

Indexation

Il est possible de créer des indexes à l'aide de l'instruction ensureIndex({champ:1}) sur une collection donnée. La valeur 1 peut être remplacée par -1, indiquant le sens de l'indexation (asc vs desc). L'instruction explain() va nous permettre d'observer l'impact de l'utilisation d'un index lors de l'exécution d'une requêt : db.collection.find(..).explain()

13. Ecrire une requête permettant d'obtenir les documents dont le score est 30. Lancer la même requête avec un explain()
14. Ajouter un index à collection ex sur la clé score et lancer la requête précédente avec un explain(). Comparez les résultats.
15. Il est possible de créer un index sur les clés d'un tableau. Créer un index sur qte dans le tableau items. Lancer une requête affichant les produits des items dont la quantité correspond à une valeur existante dans la base. Etudier le résultat d'un explain sur cette requête.

MapReduce

La solution map reduce s'inspire de l'approche du framework du même nom de Google mais ne s'intègre pas directement avec ce dernier. Il y a des points en commun : elle peut être parallélisée sur plusieurs serveurs MongoDB, elle débute avec une étape de map sur chaque document de la collection , ensuite intervient une étape (shuffle) où les clés sont groupés pour émettre des listes de couples clé/valeur. Enfin l'étape de reduce utilise ces listes pour calculer un élément à partir des valeurs stockés. Cet élément est ensuite renvoyé au shuffle jusqu'à ce qu'un seul élément (le résultat) soit calculé.

Dans MongoDB, MapReduce peut être utilisé pour calculer les opérations de count, distinct, group mais de manière moins efficace en performance que les opérations du framework d'agrégation. Les fonctions de map et reduce sont écrites en javascript.

N.B. : Il existe tout de même un connecteur Hadoop pour MongoDB

On se base sur l'exemple suivant :

```
{_id:1,favs:["spurs","knicks"]}
{_id:2,favs:["lakers","spurs"]}
{_id:3,favs:["knicks","celtics"]}
```
On veut compter le nombre d'occurences dans le tableau `favs`.

16. Saisir le jeu d'essai dans la base esipe2 et la collection basket
17. On va écrire une fonction javascript qui va émettre un couple clé/valeur pour chaque valeur rencontrée dans le tableau « favs » :
    map = function() {
        this.favs.forEach(function(f) {emit(f, {count:1});});}
    Avant de poursuivre, on peut tester. Pour cela, on va saisir le fonction emit (qui est implémentée dans la fonction mapReduce) :
    function emit(k, v) {
      print("emit");
       print("k:" + k + "v:" + tojson(v));
    }
    et on tape : map.apply({db.basket.findOne()})
18. Maintenant que l'on a vu le principe de la fonction map, on va écrire la fonction reduce. Celle-ci prend en entrée des couples clé/valeurs et retourne une clé/valeur.
    Dans notre exemple, la fonction reduce sera la suivante :
    ```
    reduce = function(key, values) {
       total = 0;
       for (var i=0;i<values.length;i++) {
        total += values[i].count;
      }
      return {"count" : total};
    }
    ```
    On peut tester localement : `reduce("celtics",[{count:1},{count:2}])`
19. On va maintenant tester au sein du framework mapreduce en utilisant nos fonctions map et reduce : res= db.basket.mapReduce(map,reduce,{out:{inline:1}})
    l'option out peut prendre plusieurs valeurs. Ici inline:1 indique qu'aucune collection ne sera créée à la suite de l'opération et donc que le résultat sera stocké en RAM (le résultat doit faire moins de 16Mo). Autres options : {replace : "nomCollection"]} pour stocker le résultat dans une collection, merge (si recouvrement de clés, al nouvelle valeur écrase l'ancienne et reduce (relance un reduce si dans l'ancien collection, il existait un clé identique)
    On pouvait aussi utiliser la syntaxe :
    db.runCommand({mapreduce :"basket","map":map,"reduce":reduce})

20. Fournir en utilisant l'approche mapReduce le nombre d'occurences par score dans la collection ex de la base de données esipe2.
21. Vérifier le résultat avec une requête exploitant le framework d'agrégation

Réplication

Il existe plusieurs solutions pour la réplication sur mongodb. Réplication Master-Slave ou au sein d'un replica set (un sur-ensemble de Master-slave). Elles sont asynchrones. On se concentre sur la réplication Master-slave.

22. Sur 2 machines ou 2 consoles sur une même machine : M1 et M2 :
    M1 : Création master :
    mkdir -p ~/dbs/master
    ./mongod --dbpath ~/dbs/master --port 10000 --master
    M2 :Création slave
    mkdir -p ~/dbs/slave
    ./mongod --dbpath ~/dbs/slave --port 10001 --slave --source localhost:10000
    Sur M1 et M2, lancer mongo.
    Créer un document sur un collection xy dans le master. Vérifier l'état de M2.
23. Ajouter un autre slave du master sur le port 10002.
24. Un replica set est un cluster Master-Slave avec une gestion de dépannage automatique (automatic failover). C'est-à-dire que le replica set va élire un master sur le cluster mais si

celui-ci tombe en panne, une autre machine du cluster prendra sa place (ce mécanisme n'est pas disponible dans l'approche Master-Slave classique). Terminologie : le master est le primary node et les slaves sont secondaries. La configuration d'un replica set est similaire à la config de la question 22 mais aucune spécification du rôle (master ou slave) n'est nécessaire et on ajoute la commande –replSet. On donne un nom au replica set (par exemple rsIRIG3) et chaque nœud se voit indiquer les autres membres du replica set.
Ex : ./mongod –dbpath ~/dbs/node1 –port 10001 –replSet rsIRIG3/localhost:10002
Idem pour le nœud au port 10002 en indiquant la présence localhost:10001.
Enfin, il faut initialiser le replica set sur une machine du cluster, par exemple sur localhost:10001 : ./mongo localhost:10001/admin
db.runCommand({replSetInitiate: {
{_id : ''rsIRIG3",
''members'' : [{ _id:1, host : localhost:100001''}, {..}]}}})
Mettre en place un replica set sur un groupe de 3 machines.
Créer une collection et ajouter un document sur celle-ci
Vérifier l'état des autres nœuds du cluster.
Utiliser le mode d'administration pour voir l'état des différents nœuds.

Sharding

Le sharding est le processus permettant d'éclater les données et de les stocker sur différentes machines. C'est la principale méthode pour permettre le scale out. La difficulté du sharding est la décomposition des données. MongoDB supporte une approche automatique : autosharding. Pour cela, on lance un serveur mongos et on indique la clé de sharding. Bien souvent, la clé de sharding correspond à un index.

25. On lance un mongod sur un port donné (par exemple 20000).
On lance un mongos sur le port 30000 avec la commande –configdb localhost:20000
Pour ajouter un nouveau shard, on lance un autre mongod (port 10000)
puis on lance mongo sur le port de mongos avec l'instruction admin :
mongos localhost:30000/admin et on ajoute le serveur du port 10000 :
db.runCommand({addshard :''localhost:10000'', allowLocal : true})
allowLocal est nécessaire sur le shard tourne sur le localhost.
On configure en indiquant la BD et la collection qu'il faut partitionner. Par exemple, on veut sharder la BD db1, collection toto sur _id. :
db.runCommand({''enablesharding'' :''db1''})
db.runCommand({shardcollection '' :''db1.toto''.key:{''_id'':1}})
Les données stockées dans la BD sont maintenant partitionnées sur différentes machines.

En production, on utilise replica set avec sharding : chaque shard a son replica set. De cette manière si un nœud tombe, l'ensemble du système ne tombe pas.