
mongoDB.

Introduction

- Une base de données orientée **document**
- Open source
  - Sponsorisé par 10gen, licence commerciale
- Haute performance
- Passage à l'échelle de manière horizontale
- Nombreuses fonctionnalités
- Première version en 2009

Document

- Un document est un tableau associatif typé
  - PHP Array
  - Python Dict
  - Ruby Hash
  - **Objet JSON**
  - etc.

Haute performance

- Ecrit en C++
- Données sérialisées en BSON
- Support pour indexes primaire et secondaire
- Modèle document impose moins de tâches d’I/O
- Moteur de stockage
  - Memory-mapped files
  - WiredTiger

Fonctionnalités

- Langage de requête
- Agrégation
- Géospatial
- Support pour de nombreux langages de prog.
- Flexibilité des schémas
- Réplication

Terminologie

SGBDR	MongoDB
Table, view	Collection
Row	Document
Index	Index
Jointure	Document imbriqué
Clé étrangère	Reference
Partitionnement	Shard

Compromis de MongoDB

- Abandon des jointures
- Abandon des transactions ACID
- Abandon de la notion de schéma
- Abandon de la durabilité sur une seule machine

Modélisation

- Modélisation dans une base de données orientée document tient compte
  - Du rapport entre lecture et écriture sur les éléments de la base
  - Taux de modification des éléments sur lesquels les modifications sont effectuées.
  - Taux de croissance du document

Modélisation

- Les éléments précédents vont justifiées :
  - La décomposition en différentes collections
  - La création de sous-document dans des documents existants.

Langage de requêtes

Présentation

- On écrit les requêtes dans le shell de MongoDB (mongo) sous forme de scripts Javascript
  - On utilise une BD avec *use dbName*, on obtient la liste de bd avec *show dbs*
  - On obtient la liste de collections d'une base de données avec *show collections*
  - Utilisation de l'approche orientée objet de JS
    - db.nomCollection.operation(…)

Récupération de données

- Récupération d’un document
  - db.maCollection.findOne()
  - db.maCollection.find()
- Ajouter *.pretty()* en fin de requête pour avoir un affichage correctement formaté

Récupération de données

- Ajouter des filtres comme paramètre de l'opération find()
  - db.maCollection.find({attribut : valeur, ...})
- N'obtenir que certaines valeurs d'un document retourné
  - db.maCollection.find({attribut : valeur},{attribut1 : valeur2, ...})
    - Avec spécifiant l'affichage dans le résultat : valeur2 = 0 (false) ou 1 (true)

Sélecteurs

- Comparaison

  $eq Matches values that are equal to a specified value.

  $gt Matches values that are greater than a specified value.

  $gte Matches values that are greater than or equal to a specified value.

  $ltMatches values that are less than a specified value.

  $lte Matches values that are less than or equal to a specified value.

  $ne Matches all values that are not equal to a specified value.

  $in Matches any of the values specified in an array.

  $nin Matches none of the values specified in an array.

- Exemple

  db.maCollection.find({annee : {$eq : "2017"}})

  db.maCollection.find({annee : {$in : ["2016", "2017"]}})

Sélecteurs

- Logique

  $or Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

  $and Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.

  $not Inverts the effect of a query expression and returns documents that do not match the query expression.

  $nor Joins query clauses with a logical NOR returns all documents that fail to match both clauses.

- Exemple

  db.maCollection.find({$or : [{gender : "Male"},{age : "27"}] })

  db.maCollection.find({annee : {$not: {$lte : "2016"}}})

Sélecteurs

- Element

  $exists Matches documents that have the specified field.

  $type Selects documents if a field is of the specified type.

- Exemple

  db.maCollection.find({annee : {$exists : 1})

  db.maCollection.find({annee : {$type : 2})

  - Type 2 est String. Cf. https://docs.mongodb.com/manual/reference/operator/query/type/#op._S_type
    pour une liste des types

Sélecteurs

- Evaluation

  $mod Performs a modulo operation on the value of a field and selects documents with a specified result.

  $regex Selects documents where values match a specified regular expression.

  $textPerforms text search. (nécessite un index)

  $whereMatches documents that satisfy a JavaScript expression.

- Exemple

  db.maCollection.find({annee : {$mod : [divisor, remainder]}})

  { <field>: { $regex: /pattern/, $options: '<options>' } }

  { <field>: { $regex: 'pattern', $options: '<options>' } }

  { <field>: { $regex: /pattern/<options> } }

  db.movies.find({title : {$regex: ‘iolence’}},{title:1, year:1})

Sélecteurs : $text

- db.articles.createIndex({ subject: "text" })
- db.articles.find({ $text: { $search: "coffee" } })

Sélecteurs sur les tableaux

- $all Matches arrays that contain all elements specified in the query.
- $elemMatch Selects documents if element in the array field matches all the specified $elemMatch conditions.
- $size Selects documents if the array field is a specified size.
- Exemple
  - db.maCollection.find({drinks : {$all : ["Coffee","Tea"]}})
  - db.maCollection.find({drinks : {$size : 2}])
  - db.maCollection.find({personnes : {$elementMatch : {prenom : "Marie"}}})

Projections

- $ Projects the first element in an array that matches the query condition.
- $elemMatch Projects the first element in an array that matches the specified $elemMatch condition.
- $meta Projects the document’s score assigned during $text operation.
- $slice Limits the number of elements projected from an array. Supports skip and limit slices.
- Exemple
  - db.maCollection.find({annee : {$eq : 2005}},{"personnes.$":1})
  - db.maCollection.find({annee : {$eq : 2005}},{personnes : {$slice:2}})

Insertion de documents

- A l'aide de db.maCollection.insertOne({ ..})
  - Insertion d'un unique document
- db.maCollection.insertMany([d1,d2])
  - Insertion d'un ensemble de documents
- Ou bien db.maCollection.insert({ .. })
  - Pour 1 ou plusieurs documents
- On peut aussi mettre un variable JS comme paramètre

Insertion de documents

- d1 = {name:"a",tags:["a","b","c"]}
- d2 = {name:"b", tags:["b","c","d"]}
- db.test.insertMany([d1,d2])
- db.test.find()
- db.test.count() // retourne 2

Agrégation

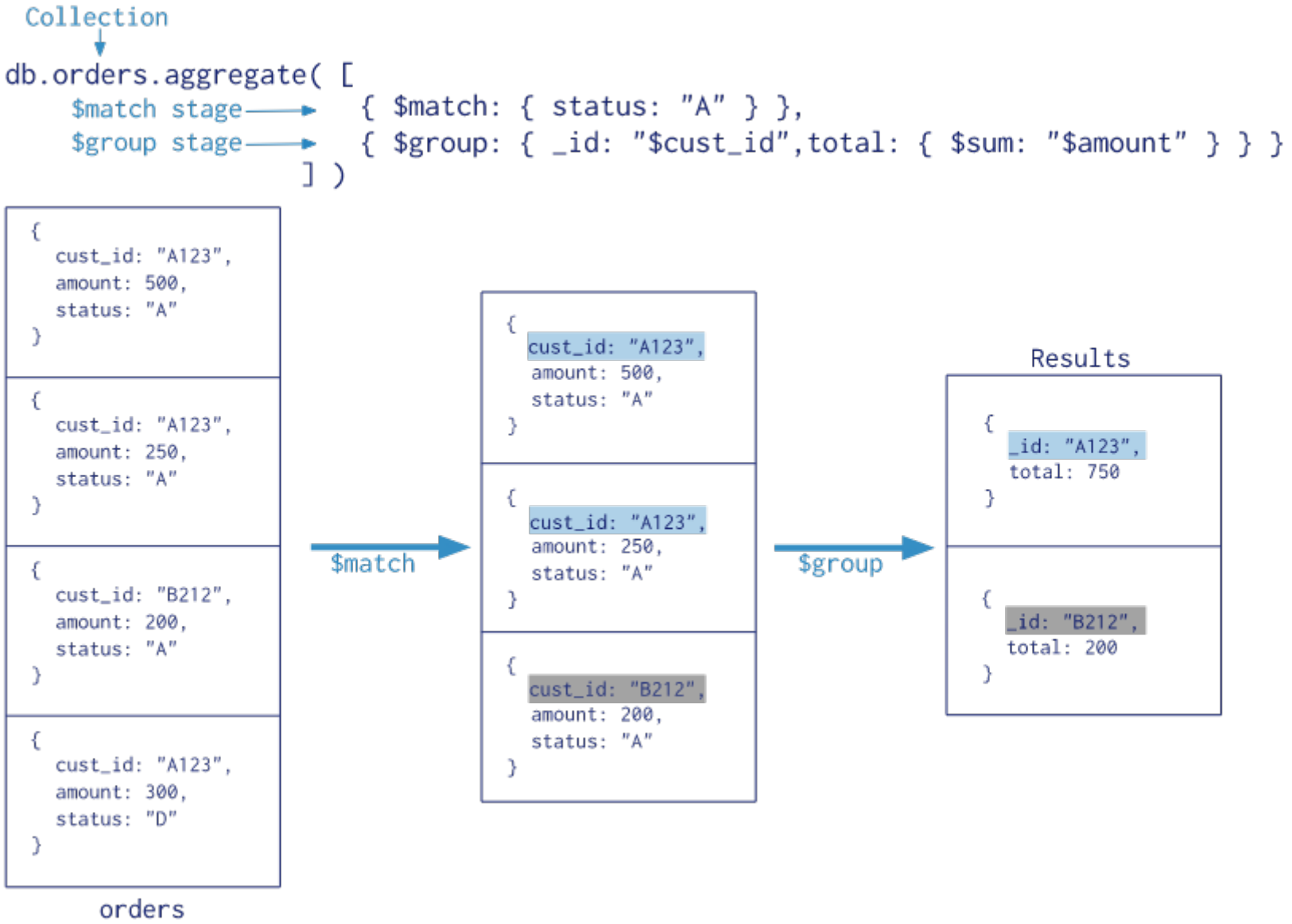
- Compter le nombre d’éléments : db.coll.count()

- Obtenir des valeurs distinctes : `db.collection.distinct(field, query)`
  - Query filter les champs
  - Fonctionne avec des sous documents et des listes

Agrégation (2)

- Le framework d’agrégation s’inspire des chaînes de traitements des moteurs de big data (e.g., Spark, Flink, etc.)
- Les documents rentrent dans une suite de traitements s’exécutant les uns après les autres jusqu’à obtenir un agrégat.
- On y exécute des filtres et des transformations

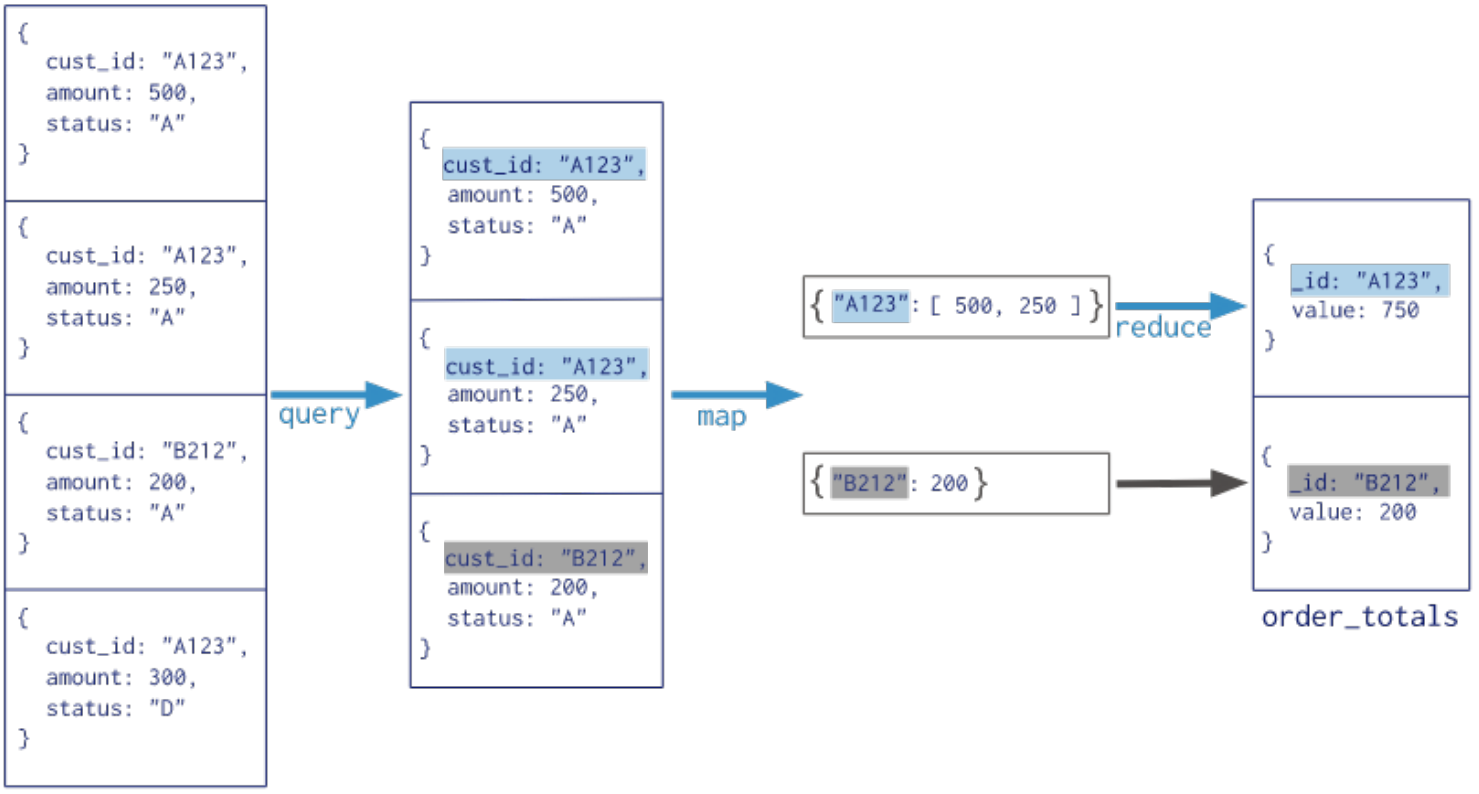
Agrégation (3)



Agrégation (4) : Map Reduce

Collection

```
db.orders.mapReduce(
        map      ──►  function() { emit( this.cust_id, this.amount ); },
        reduce   ──►  function(key, values) { return Array.sum( values ) },
                      {
        query    ──►    query: { status: "A" },
        output   ──►    out: "order_totals"
                      }
                    )
```



Indexation

_id

- La clé primaire dans MongoDB
- Indexation automatique, par défaut
- Automatiquement créé comme un ObjectIf si non donné au moment de la création d’un document
- Une valeur unique non mutable est un bon candidat

ObjectId

- Une valeur correspondant à 12 octets
  - 4 octets pour l’estampillage
  - 3 octets pour l’id de la machine
  - 2 octets pour l’id du processus
  - 3 octets pour le compteur
- Garantie d’unicité

Indexation secondaire

- MongoDB permet la définition d'indexe secondaire sur des éléments JSON d'un document
- Création à l'aide de `createIndex` (`ensureIndex` est déprécié):
  - db.collection.createIndex({attribut : 1})
    - Avec 1 pour un tri en ordre croissant ou -1 pour un ordre décroissant
- On peut également créer un indexe sur une structure imbriquée
  - db.collection.createIndex({sousDoc.attribut : 1})

anciennement ensureIndex

Indexation

- On peut indexer l'ensemble des attributs d'un sous document (*multikeys*)
  - `db.collection.createIndex({sousDoc:1})`
    - Dans ce cas tous les attributs du sous document sont indexés et ordonnées dans l'ordre croissant
- On peut déclarer des indexes avec plusieurs clés (*compound keys*):
  - `db.collection.createIndex({sousDoc.att1:1, sousDoc.att2 :-1})`

Indexation

- Egalement possible avec les éléments d’un tableau
  - db.movies.createIndex({actors.name:1})

hint

- On peut forcer l'usage d'un index à l'aide de la fonction *hint* :
  - `db.collection.find( { att1 : valeur}) . hint ( { att1: -1 } )`
  - L'index doit exister sur l'attribut sinon le système retourne une erreur
- Comme dans un SGBD, on a accès à une opération explain pour voir le plan d'exécution d'une requête :
  - `db.collection.find({..}).explain()`

Création d’un indexe

- Par défaut, la création d’un indexe bloque toutes les autres opérations (lecture/écriture) sur la base.
- Pour pallier à ce problème on peut indexer à l’aide de l’opération `background` :
  - db.collection.createIndex({att1:1},{background:1})

Min/max

- Si on veut utiliser les opérations min et max, il faut avoir indexer préalablement sur l'attribut :
  - `db.collection.find() . min ( { att: 1995 } ) . max ( { att : 2005 } )`

Gestion des indexes

- On peut obtenir les indexes d'une collection avec `getIndexes()`
  - db.collection.getIndexes
- La liste de tous les indexes d'une bd avec `listIndexes()`

Administration

Ce que l’on va voir

- Sauvegarde et restauration d’une BD Mongo
- Tâches courantes sous le shell mongo
- Contrôle d’accès avec authentification
- Observation des performances d’instances

Les outils

- Certains sont inclus lors de l'installation
  - Ils se trouvent dans le répertoire *bin/* de l'installation
- D'autres sont disponibles sur le Web (third-party tools) :
  - https://docs.mongodb.com/ecosystem/tools/
  - http://mongodb-tools.com/
- On se concentre sur les outils inclus dans MongoDB

Sauvegarde

- On sauvegarde une base de données MongoDB à l'aide de l'exécutable *mongodump*.
- Le serveur doit être lancé (mongod)
- On va créer un répertoire où les dumps seront sauvegardés
- Dans ce répertoire, on invoque mongodump (qui se trouve dans bin/)

Sauvegarde

- Des options sont disponibles
  - On y accède en lançant mongodump –help
  - En particulier, on peut
    - Identifier une bd, une collection
    - Renseigner l’authentification
    - Etc.

Sauvegarde

- A la suite d’une exécutuion de mongodump
  - On obtient un répertoire dump dans le répertoire spécifié
  - Celui-ci contient un bson par collection et un fichier JSON de métadonnées pour chaque collection

Restauration

- A partir d’un dump préalablement créé avec mongodump
- La commande mongorestore doit être lancée depuis le répertoire du dump
- Plusieurs options :
  - Indication de la BD, collection
  - Authentification
  - Option ‘--drop’ pour tout supprimer avant la restauration

Importation de données

- *mongorestore* restaure depuis une suavegarde
- ***mongoimport*** est un exécutable permettant d'importer des données depuis des sources
  - CSV
  - TSV
  - JSON
- Ce programme présente de nombreuses options qui peuvent être obtenues avec un mongoimport --help

Gestion des utilisateurs

- Cela se fait depuis la console MongoDB
- On ajoute des utilisateurs puis on active l'authentification
  - `use admin`
  - `db.createUser({user:'admin',pwd:'pass', roles: [{role:'readWrite',db:'admin'}, {role:'userAdminAnyDatabase',db:'admin'}]})`

Gestion des utilisateurs

- Rôles
  - `read`
  - `readWrite`
  - `dbAdmin` : lancer des opérations d'administration, ex : création, suppression d'indexes, voir des stats
  - `userAdmin` : permet la création, suppression, administration d'utilisateurs pour cette bd
  - `dbOwner` : une combinaison de readWrite, dbAdmin et userAdmin
  - `clusterManager` : que pour la base admin
  - `clusterMonitor` : que pour la base admin, accès aux stats et commandes pour rassembler les stats.

Gestion des utilisateurs

- Rôles
  - `readAnyDatabase` : autorise la lecture sur toutes les bases du serveur
  - `ReadWriteAnyDatabase`
  - `userAdminAnyDatabase`
  - `dbAdminAnyDatabase`

Gestion des utilisateurs

- Activation de l'authentification
  - Au moment du lancement du serveur, on ajoute *--auth* en fin de ligne
  - On lance ensuite mongo, on utilise la base admin et on saisit : db.auth('admin','pass')
  - On peut ensuite obtenir la liste des bases, collections, requêter, etc..

Gestion des utilisateurs

- Ajouter un utilisateur en lecture seule
  - `use admin, db.auth(..), use dbcible, db.createUser({user:'toto',pwd:'toto', roles: [{role:'read',db:'dbcible'}]})`

Gestion des utilisateurs

- Supprimer un utilisateur
  - `use admin, db.auth(..), use dbcible, db.dropUser('toto')`

Gestion des utilisateurs

- Modifier les droits d'un utilisateur
  - `use admin, db.auth(..), use dbcible, db.updateUser('toto',{roles: [{role:'dbAdmin',db:'dbcible'}]})`

Gestion des logs

- Les logs de MongoDB sont par défaut affichés dans la console du mongod (vérifiez)
- On peut les diriger vers un fichier avec l'option --logpath
- C'est bien utile si on veut étudier les messages relatifs aux pannes.
  - `mongod --logpath myLogFile`

Statistiques

- L’exécutable mongostat pour obtenir une vue globale sur ce qui se passe sur un serveur.

Réplication

Pourquoi répliquer

- Améliorer la montée en charge (*scalability*)
  - Amélioration de la redondance
    - Sur plusieurs data centers → localité des recherches
  - Amélioration de la performance
    - Parallélisation des requêtes
- Améliorer la durabilité/fiabilité
  - Pour une reprise sur panne
  - Redondance pour raison administration : rotation avec serveur en production

Grands principes

Replica set

- Un Replica set correspond à un cluster maître/esclave avec reprise sur panne automatique
- Il y a un maître (primary) et plusieurs esclaves (secondary)
- Un replica set nécessite un majorité de membres votant pour maintenir un primary
  - Il faut au moins 3 membres et de préférence un nombre impair de membres (au max env. 50)

Replica set : primary

- Le primary est la référence de l'état d'un replica set.
- Le primary est le seul nœud pouvant recevoir des écritures depuis une app et celui depuis lequel les secondaries sont synchronisés
- Il est élu par les membres suivant un vote par majorité
- Etre primary est éphémère

Replica set : secondary

- Un secondary stocke une copie des données
- Il peut être lu (attention à la méthode d’accès)
  - La réplication est asynchrone. Donc attention car les esclaves peuvent ne pas voir les dernières mises-à-jour
- Il peut devenir primary si le primary tombe et que les membres l’élisent.

Oplog

- Le master stocke ses opérations dans un operation log (oplog)
- Les esclaves se synchronisent avec le master en exploitant le oplog
- Les membres d’un replica set stockent également un oplog
- Une réplication en chaîne, de secondary en secondary est possible.

Oplog (2)

- Le oplog va créer une estampille pour chaque nouvelle entrée
- Un secondary peut donc vérifier à partir de cette estampille s'il faut considérer une nouvelle entrée
- Oplog est une (capped) collection pour ne pas garder de collections trop volumineuses
  - Taille configurable (entre 1Go et 50Go)

Replica set

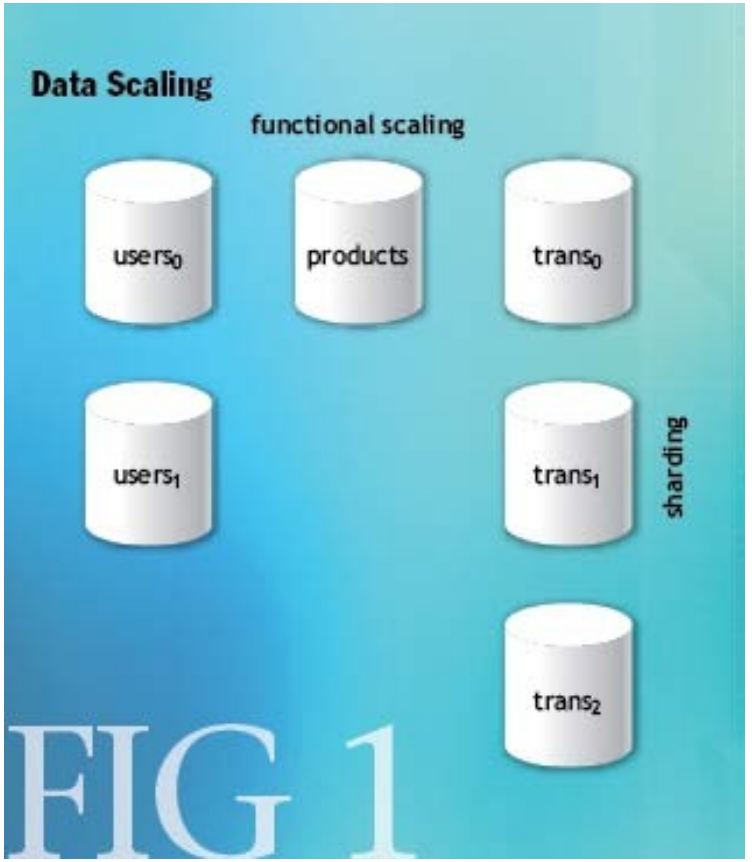
- Heartbeat pour la communication entre nœuds
- Consistance forte
- Primary est le seul nœud à prendre des écritures

Sharding

Objectif

- **Scalability** (montée en charge)
  - La capacité d'un système, réseau, processus à gérer un volume croissant de tâches d'une manière efficace ou sa capacité à s'élargir pour s'adapter à cette croissance
  - 2 approches
    - Vertical scaling (scale up) : on ajoute de la mémoire, des disques, des CPU
    - Horizontal scaling (scale out) : avec du **functional scaling** où on regroupe les données par fonction et on distribue celles-ci sur un ensemble de machines et/ou du **sharding** où on découpe les données au sein des fonctions sur plusieurs machines.

Scaling out


Data Scaling
functional scaling
users0
products
trans0
users1
trans1
sharding
trans2
FIG 1

Principe

- La solution de MongoDB pour faire du scale out
- Principe : on découpe les collections en morceaux qui sont distribués sur des shards.
- L'exécutable prenant en charge le sharding est *mongos*.
  - Il s'occupe de la distribution des requêtes
  - L'utilisateur se connecte à mongos à la place de mongod
  - Le reste est **transparent** pour l'utilisateur
    - On a l'impression d'interagir avec une unique BD

Passage à l’échelle

- Utilisateur définit une clé pour le sharding
- Shard keys vont définir des intervalles (ranges)
- Les intervalles sont répartis sur les machine du cluster
  - Par exemple si la clé est le nom d’un utilisateur et que l’on dispose de 2 machines, on peut avoir le utilisateurs dont les noms débutent par A à M sur la machine 1 et de N à Z sur la machine 2.
- MongoDB gère l’équilibrage des shards.
  - Peut-être beaucoup plus d’utilisateurs sur A-M que sur N-Z.

Effets du sharding

- Avec du sharding, on baisse les chances de non disponibilité des données car on ne dispose plus d’une seule machine mais de plusieurs
- On augmente le nombre de machines et donc les chances d’avoir des pannes des machines.
- Pour avoir une architecture fiable, on combine souvent sharding et réplication

Maintenance à chaud

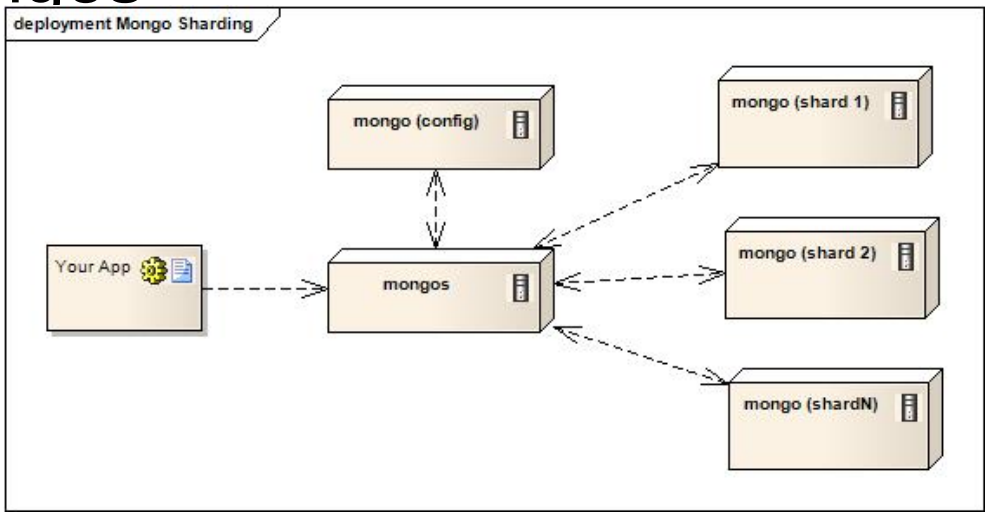
- Une bonne solution de sharding permet d’ajouter et de retirer des machines d’un cluster sans interruption de services et sans avoir à faire de sauvergarde et/ou restauration de manière manuelle.

mongos

- L’exécutable mongos sert de contrôleur sur un ensemble de serveurs de shard basés sur mongod.
- Une application se rattache à un mongos comme si elle se connectait à un unique serveur.
  - Toutes les commandes sont alors envoyées (ex. requêtes) sur le mongos

mongos

- L'exécutable mongos sert de contrôleur sur un ensemble de serveurs de shard basés sur mongod.
- Une application se rattache à un mongos comme si elle se connectait à un unique serveur.
  - Toutes les commandes sont alors envoyées (ex. requêtes) sur le mongos



Requêtage

- A la réception d’une requête, mongos va distribuer son exécution et gèrera la création d’un résultat final (par agrégation des résultats intermédiaires)
- La granularité du sharding est au niveau de la collection, pas de la BD
  - On peut donc partitionner un sous-ensemble des collections d’une BD sans partitionner les autres

Config Server

- Un config server est utilisé dans le cas du Sharding MongoDB
  - Il stocke des métadonnées sur le cluster, par exemple où se trouve un ensemble de documents identifiés par un intervalle de la clé de sharding
  - C’est un mongod avec un rôle spécifique
- On peut faire du sharding sans avoir un grand nombre de machines physiques. On peut co-héberger des serveurs sur une même machine

Shard key

- Choisir une clé utilisée dans des requêtes
- Elle est non mutable
- Elle doit être indexée
- Sa taille est limitée

Critères sur Shard key

- Cardinalité
- Distribution des écritures
- Isolation des requêtes
- Distribution des données

Quand faire du sharding ?

- Problème d’espace disque sur la machine courante
- Besoin d’écrire plus rapidement
- Besoin de mettre plus de données en mémoire vive

GridFS

GridFS

- Par défaut, la taille d'un document stocké par MongoDB est limitée à 16Mo
  - Essentiellement pour garantir la performance
- Dans certaines applications, cela peut vite devenir une limitation importante
- GridFS permet de pallier à cette limitation. GridFS est simplement une nouvelle méthode de stockage des fichiers dans MongoDB

Usages

- Lorsque d’un document dépasse la taille de 16Mo
- Lorsque vos besoins ne nécessitent pas le chargement complet d’un document. Par exemple, je veux voir les n premières pages d’un gros document. Je veux accéder directement au kième chapitre d’un ouvrage
- Ne pas utiliser GridFS si on a besoin de faire des modifications atomiques sur un fichier

Principe de GridFS

- GridFS comporte 2 collections :
  - Une collection (*files*) stocke le nom de fichier et des métadonnées (taille)
  - Une collection (*chunks*) contient les données, découpées en morceaux de 255Ko (configurable mais pas plus de 16Mo)
- Ces collections sont stockées dans l'espace de données *fs* (mais c'est modifiable, par exemple pour séparer des images et des vidéos).