Notas sobre mongoDB Documentation

Release doc 1.0

yorsh

January 20, 2015

CONTENTS

CHAPTER

ONE

INTRODUCCIÓN

Características sobre MongoDB

- No JOINS
- Orientado a documentos (agrupación de ditintos campos)
- Existen varios tipos de datos declarados en el documento y cada atributo puede ser de distinto tipo en cada documento
- Preparado para alto rendimiento. Intenta mapear lo mas posible en memoria
- Limitaciones
  - En los sistemas de 32bit tiene una limitación de 2GB
  - En Windows < 8 tiene una limitación de 4TB
  - Linux x64 alcanza hasta 64TB
- Usan BSON (http://bsonspec.org) para almacenamiento. Formato binario de JSON.
- Escalabale horizontalmente.
  - Shard (particiones). No tienen porque estar en el mismo CPD
- Mas features que otras BD NoSQL
  - rich queries
  - geoespacial
  - text search
  - agregación
  - map reduce

Documentos

- Un documento no puede estar fragmentado en disco
- El diseño para alto rendimiento está pensado en la forma de presentar los datos en la pantalla
- El **tamaño límite** de un documento son 16MB
- Los ficheros de metadatos (ej.: local.ns) pueden ocupar hasta 16MB, esto equivale a unos 24000 namespaces

---

Equivalencia de conceptos SQL

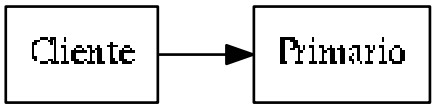
SQL	MongoDB
Table, view	Collection
Row	Document
Index	=
Join	Embedded Document
FK	Reference
Partition	Shard

CHAPTER

TWO

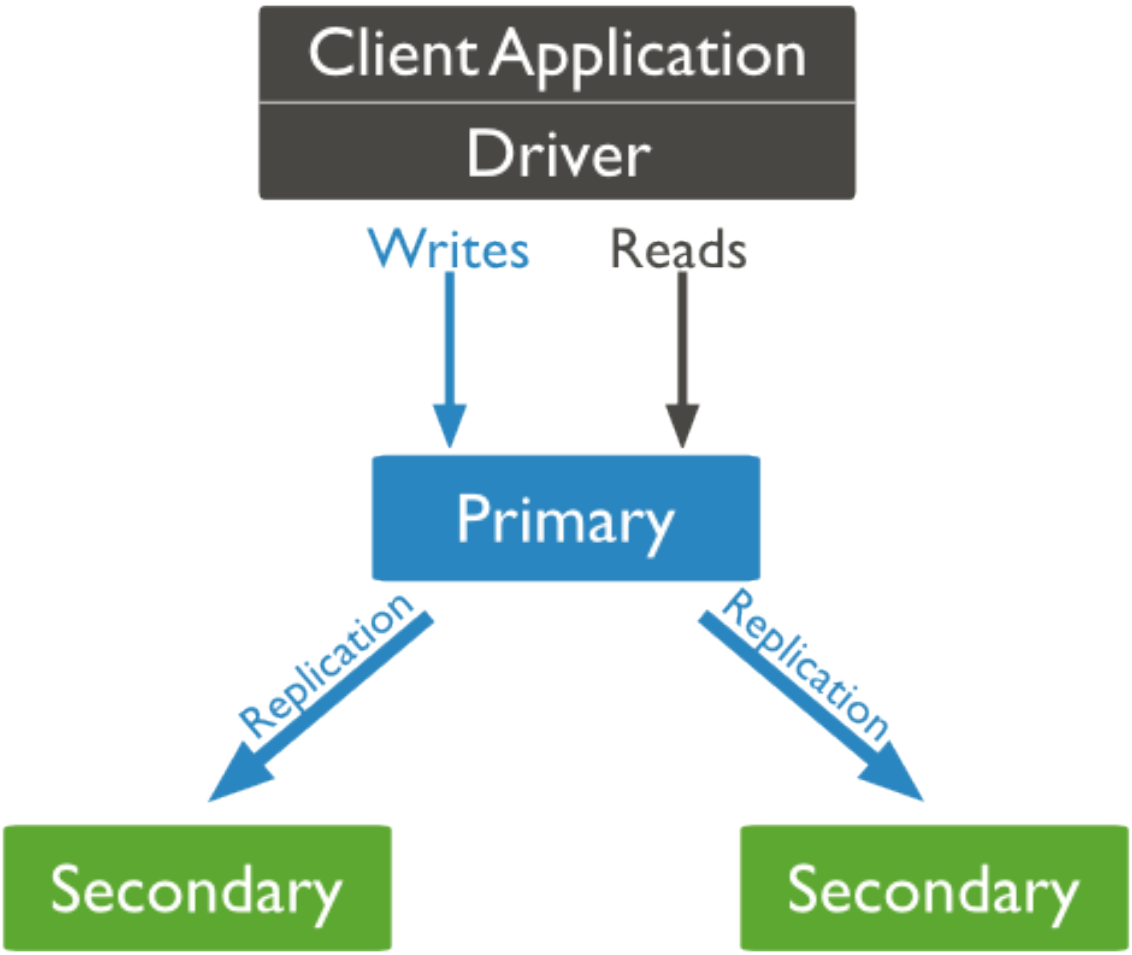
DESPLIEGUES TIPO

2.1 Standalone



2.2 Replica Set

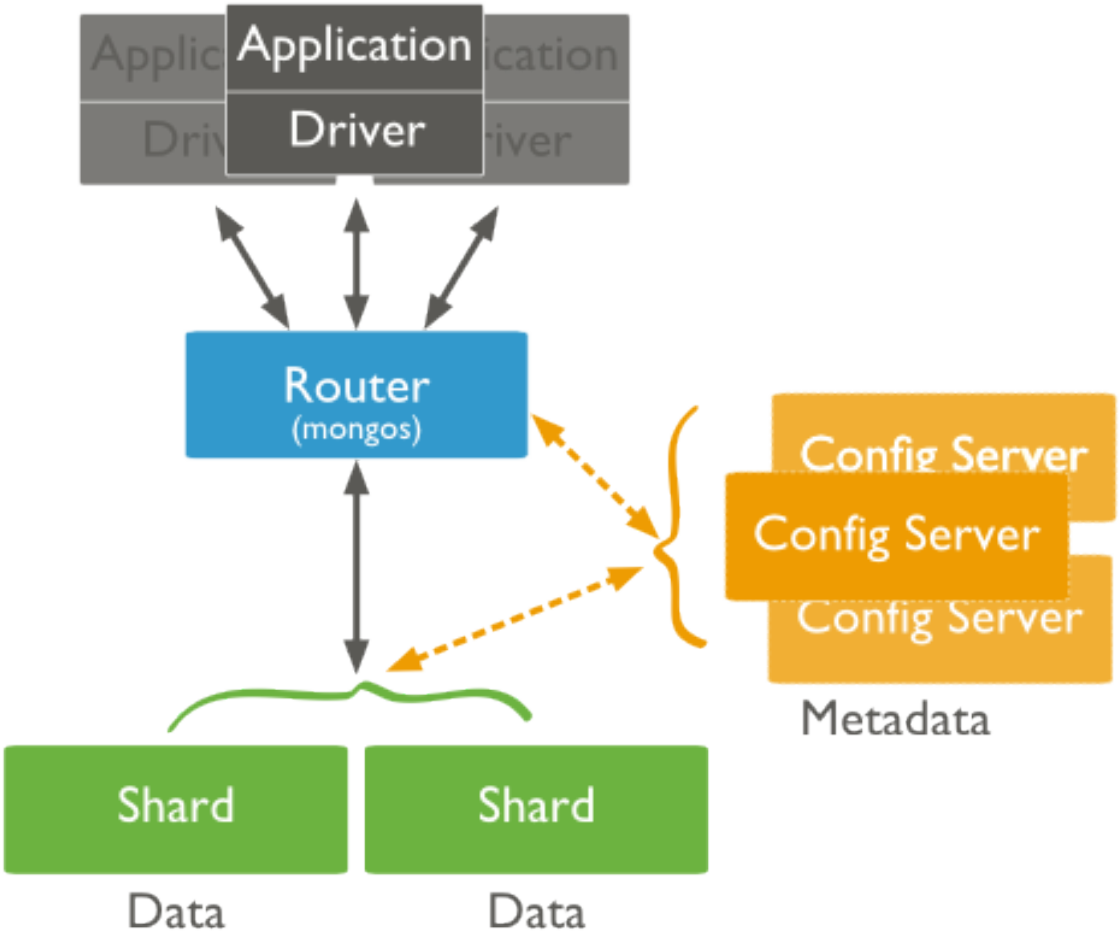
- Máximo 1 primario
- Máximo 11 secundarios
- Escrituras solo en primario
- Se pueden hacer lecturas conectándose a un secundario



2.3 Sharded cluster

A través del config server se sabe a que replicaSet (shard) se puede escribir/leer Config servers posibles:

- 1 - Usado para pruebas
- 3 - Usado para producción. Provee tolerancia a fallos


Application
Driver
Router
(mongos)
Config Server
Config Server
Config Server
Metadata
Shard
Shard
Data
Data

CHAPTER

THREE

LÍNEA DE COMANDOS

mongod primary daemon process for the MongoDB system

mongos "MongoDB Shard," routing service for MongoDB shard configurations

mongo interactive JavaScript shell interface to MongoDB

mongodump utility for creating a binary export

mongorestore writes data from a binary database dump created by mongodump to a MongoDB instance

bsondump The bsondump converts BSON files into human-readable formats, including JSON

mongooplog simple tool that polls operations from the replication oplog of a remote server, and applies them to the local server

mongoimport import content from a JSON, CSV, or TSV export created by mongoexport

mongoexport produces a JSON or CSV export of data stored in a MongoDB instance

mongostat quick overview of the status of a currently running mongod or mongos instance

mongotop method to track the amount of time a MongoDB instance spends reading and writing data.

mongosniff low-level operation tracing/sniffing view into database activity in real time

mongoperf check disk I/O performance independently of MongoDB

mongofiles makes it possible to manipulate files stored in your MongoDB instance in GridFS objects from the command line

3.1 mongod

```
[--dbpath]                          default= /data/db, c:\data\db
[--port]                            default=27017
[--journal | --nojournal]           active by default on 64 bits
[--smallfiles]
[--nssize <MB>]                     max 2GB
[--noprealloc]
[--logpath]
[--logappend]
[--fork]                            Not supported on windows
[--config|-f <ASCII configFile>]
[--replSet <rsName>]
[--oplogSize <MG>]                  default=5% free space
[--shardsvr]
[--configsvr]
[--auth]
[--keyFile <path to file>]
[--syncdelay 45]                    seconds between disk syncs
[-vvvvv]                            Verbose level
```

Data File Allocation.

Each database will have at least two data files, one ending in .ns and the rest in integers starting with 0.

The .ns file stores metadata about namespaces (collections and indexes). The number of namespaces is proportional to the size of the .ns file. Each database can have up to 24,000 namespaces by default, although the size of these files, and thus the number of namespaces, can be increased with the `--nssize` option (up to 2GB).

By default, datafiles start at 64MB and double in size with each additional datafile, up to 2GB. Additionally, on some platforms, mongod allocates one more numbered data file than it needs, to improve throughput. Thus, it's possible for allocated size to be much larger than data size. If this presents a problem, you can use some combination of server options:

```
--smallfiles    // quarters the sizes of data files
--noprealloc    // inhibits preallocation of extra files
```

The Lock File

`mongod.lock` contains PID. When the server shuts down cleanly, it truncates the lock file to length 0.

Journal Subdirectory

Activated by default on x86_64 machines. Directory `journal`.

Note: Todos los servidores comparten el mismo journal.

Log Files

Default: standard output.

`--logpath /var/mongodb/mongodb.log --logappend` allows to write the log into a file.

The database includes a command to rotate log files:

```
db.adminCommand( { "logRotate" : 1 } ) ;
```

Config files

All of these options can be specified in a config file. Any option that takes an argument is specified as option = argument. Options that don't take arguments are specified as option = true:

```
fork = true
# vvv = true
logpath = /var/mongodb/mongodb.log
```

Since mongodb 2.6 version, a new YALM format is introduced. Example:

```
systemLog:
  destination: file
  path: /home/vagrant/mongodata/singledb/db.log
  logAppend: true
processManagement:
  fork: true
net:
  port: 27017
  http.enabled: true
storage:
  dbPath: /home/vagrant/mongodata/singledb/db
  preallocDataFiles: false
  smallFiles: true
  journal:
    enabled: true

# Otras secciones no probadas (obtenidas wiki online)
setParameter:
security:
  ...
```

```
operationProfiling:
  ...
replication:
  ...
sharding:
  ...
auditLog:
  ...
snmp:
  ...
```

3.2 mongorestore

```
mongorestore (.exe) -d digg sampledata/dump/digg
mongorestore (.exe) -d training -c scores sampledata/dump/training/scores.bson
```

3.3 mongoimport

```
mongoimport (.exe) -d twitter -c tweets sampledata/twitter.json
```

CHAPTER

FOUR

DOCUMENTOS

Ejemplo documento JSON en MongoDB:

```
{
    "_id" : ObjectId("4ba267dc238d3ba3ca000001"),
    "href" : "http://digg.com/people/Jedi_apology",
    "title" : "'Jedi' believer gets an apology!",
    "comments" : 153,
    "container" : {
        "name" : "Offbeat",
        "short_name" : "offbeat"
    },
    "submit_date" : 1268771801,
    "topic" : {
        "name" : "People",
        "short_name" : "people"
    },
    "promote_date" : 1268915964,
    "id" : "19970068",
    "media" : "news",
    "diggs" : 404
}
```

CHAPTER

FIVE

CRUD - MONGODB SHELL

Se accede con el comando `mongo`.

La base de datos por defecto es `test`.

Si se pasa la función sin paréntisis, el shell pinta la definición. Ej.: `db.startup_log.findOne`

Ejemplos comandos básicos:

```
show dbs | db.adminCommand({"listDatabases": 1})
use <dbname>
show collections | show tables | db.getCollectionNames()
db.<col>.findOne()
db.<col>.find()
db.<col>.insert(<jsonObject>)
```

Document Data Types

It's sometimes useful to see the size of an object in BSON. You can do that like so

```
Object.bsonsize( { "user.name" : "Emma Smith" } ) ;
Object.bsonsize( db.tweets.findOne( { "user.name" : "Emma Smith" } ) ) ;
```

You can also query objects by BSON type. Here's how to find all stories where href is a string.

```
use digg
db.stories.find( { "href" : { "$type" : 2 } } ) ;
```

Cargar desde un fichero:

```
load("testIndex.js")
```

5.1 Inserts

```
db.people.insert( { "name" : "Smith", "age": 30 } ) ;
```

5.2 Queries

```
db.scores.find( { "score" : { "$gte" : 70 } } ) ;
```

Operators:

- $gt / $gte / $lt / $lte
- $ne
- $in / $nin. puede ser usado tanto con números, como cualquier tipo de dato (strings, ...)
- $mod

- $regex/$options,
- $all
- $size. Consulta si el array es del tamaño que pasemos
- $exists. Comprueba si existe un campo
- $type. Comprueba si un campo es de un tipo concreto, hay que usar el número del tipo (BSON).
- $not
- $or / $nor
- $elemMatch,
- $where (try not to use $where !) **Full collection scan**. En breve quedará obsoleto y siempre tendrá valores javascript (aunque estemos desarrollando en otro lenguage)

In order to query into subdocuments, you must use "Dot notation": `db.collection.find( { "x.y" : 10 } );`

Queries and arrays

Queries that examine arrays will compare each element in the array with the argument to the matching predicate. For example, both

```
db.collection.find( { "x" : 2 } ) ;
```

and

```
db.collection.find( { "x" : { "$gt" : 4 } } ) ;
```

will match documents where x is the array [1, 2, 5]. (In the first case, 2 is present in the array; in the second case, at least one element of the array is greater than 4.)

Subset of fields (projections)

```
db.scores.find( {} , { "score" : 1 } ) ; // only includes score and _id
db.scores.find( {} , { "score" : 0 }) ; // includes everything but score
```

Cursors

To query MongoDB is to instantiate a cursor. You can also get more control using `hasNext()`, `next()` and `forEach()`

```
var cursor = db.scores.find() ;
while ( cursor.hasNext() ) printjson( cursor.next() ) ;
db.scores.find().forEach( function(x){ printjson( x ) } ) ;
db.scores.find().forEach( printjson ) ;
cursor.forEach // without () prints source
```

With `sort()`, `skip()`, and `limit()` you can control the results of the cursor

Note: el cursor tiene un tiempo de vida que suelen ser unos 10 minutos.

5.3 Updates

Danger: Without `$set` operator, full document is replaced

```
db.stuff.update( { ``_id'' : 123 } , { ``hello'' : ``world'' } ) ; // Replace full docu
db.stuff.update( { ``_id'' : 123 } ,
        { ``$set'' : { ``hello'' : ``world'' } } ) ; // Updates hello attribute
```

Operators

- $set
- $unset
- $inc
- $push
- $pushAll
- $pull
- $pullAll
- $pop
- $addToSet
- $rename
- $bit
- $ positional operator

Upserts and multi-update

`upsert`: updates or creates if not exists.

By default, only one document is updated, you must use `multi` in order to update several documents

```
db.people.update( { "name" : "Jones" } , { "$set" : { "age" : 50 } } , { "upsert" : true } ) ;
db.people.update( { } , { "$set" : { "city" : "NYC" } } , { "multi" : true } ) ;
```

CHAPTER

SIX

REPLICA SET

Sistema de votos para la elección de primarios:

- Para que un nodo pase a primario tiene que obtener la mayoría de votos (mayor que la mitad)
- Mínimo 3 nodos para poder tener recuperación automática. Sino no se puede decidir un nuevo primario.
- **Árbitro**: es un nodo secundario sin almacenamiento y solo usado para las votaciones.
- Un secundario puede tener **derecho de veto**. Cuando un secundatio tiene una fecha de actualización mayor.
- Máximo número de nodos que pueden votar: 7
  - Un nodo que no puede votar, si puede poner 1 veto.
- Los secundarios tienen prioridad, por defecto = 1.

Oplog.rs

En los nodos del replica set, existirá la colección `oplog.rs` dentro de la base de datos `local`. Esta colección se crea con las escrituras en el primario y se envía a los secundarios.

Es una colección de tamaño máximo donde se almacenan las operaciones de una forma idempotente.

Lecturas en secundarios

Hacer lecturas en los secundarios puede ser una buena idea si estamos haciendo operaciones pesadas y no nos importa que los datos no sean justo los últimos.

Al conectarse a un secundario con el comando "mongo", no nos permite lecturas por defecto, para habilitarlas debemos ejecutar:

```
rs.slaveOk()
```

Mantenimientos y upgrades.

Para mantenimientos de los servidores y actualizaciones del software, primero se deberían hacer en los secundarios.

6.1 Write concerns

- **w:0**. Mando y espero solo el ack de la red
- **w:1**. *Valor por defecto* en standalone y primarios de replica set. Cuando la escritura ha sido hecha en la memoria del servidor, aunque no en disco ni replicada.
- **w:n (siendo n>1)**. Cuando se ha confirmado en n nodos (primario y n-1 secundarios).
- **w:"majority"**. Cuando se ha confirmado en la mayoría de los miembros del replica set.
- **w:<tag set>**. Cuando se ha confirmado en los nodos configurados con el tag dado.
- **j:true**. Espera a la escritura en el journal para devolver la confirmación.

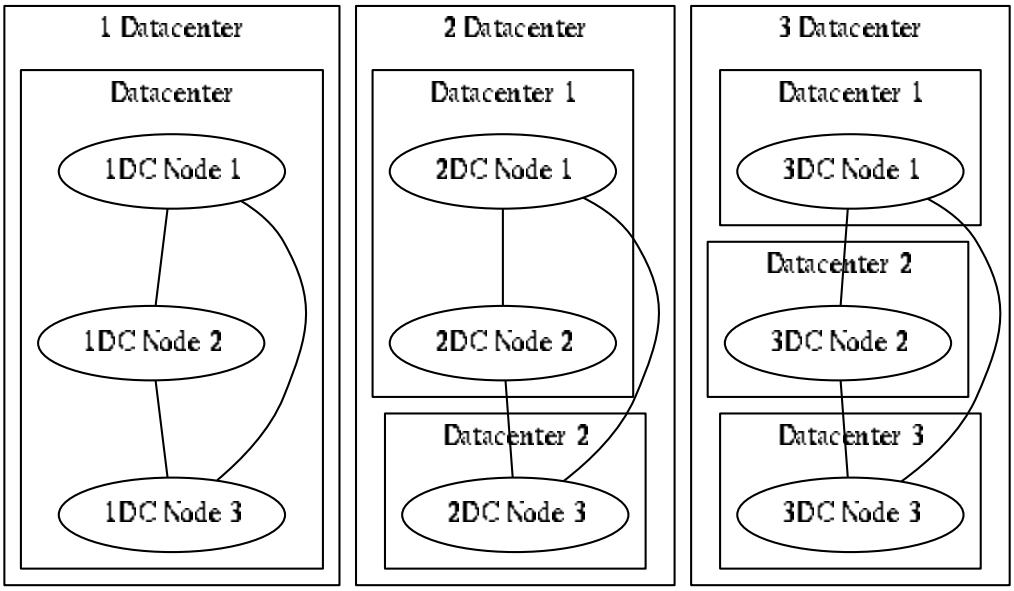
- **wtimeout:<ms>**. Timeout en milisegundos que se espera para confirmación de secundarios. Solo tiene sentido con w>1.

6.2 Read Preference modes

- **primary**. Valor por defecto. Siempre se lee en el primario
- **primaryPrefered**. Siempre en el primario a no ser que no esté disponible, en este caso se permiten secundarios.
- **secondary**. Solo se lee de secundarios.
- **secondaryPrefered**. Se lee de secundarios a no ser que ninguno esté disponible.
- **nearest**. Lee del mas cercano (nodo con menos latencia en la red) sea primario o secundario.

6.3 Datacenters

Como ya se ha dicho, para auto-recuperación de un replica set necesitamos al menos 3 nodos (si hay solo 2 y se cae el primario, el secundario no tiene votos suficientes para pasar a primario). Repartiendo los nodos en datacenters:



Podemos ver que la única configuración para poder recuperarse automáticamente ante la caída/destrucción de un data center (DR = Disaster Recovery) es usar como mínimo 3 datacenters.

Si usamos 1 y se pierde, perdemos todo.

Si usamos 2, podríamos recuperarnos en el caso de que el datacenter que se pierda sea el que contiene 1 nodo, en otro caso, no.

6.4 Configuración.

Configurando un replica set añadiendo secundarios:

- Se levantan los servidores con el nombre del replica set
  - `$ mongod ... --replSet set0`. Si se configura desde linea de comandos. "set0" es el nombre del replica set
  - `replSet=set0`. Si se configura en fichero de configuración con formato antiguo
  - `replication.replSetName: set0`. Si se configura en el fichero de configuración con formato YAML
- Nos conectamos al nodo que será el primario
  - Con `rs.status()` podemos comprobar que no está inicializado el replica set
  - Con `rs.conf()` también veremos datos de la configuración del replica set
- Con `rs.initiate()` iniciamos el replica set que solo tendrá este nodo actualmente y el shell nos debería cambiar a `set0:PRIMARY>`.
- Para añadir otros nodos `rs.add("<hostname>:<port>")`. Ejemplo: `rs.add("precise32:28002")`.
  - También es posible pasar un documento en vez de un string con la configuración del nodo.
  - Los secundarios deben ser añadidos desde el primario

Configurando un replica set pasando la configuración de todos los nodos:

Para ello, en vez de inicializar con una configuración vacía, pasamos al comando "initiate" la configuración:

```
config = { "_id": "set0", "members" : [
    {"_id" : 0, "host" : "localhost:27017" },
    {"_id" : 1, "host" : "localhost:27018" },
    {"_id" : 2, "host" : "localhost:27019" } ]
}

rs.initiate(config)
```

Valores posibles de configuración:

```
{
  _id: <string>,
  version: <int>,
  members: [
    {
      _id: <int>,
      host: <string>,
      arbiterOnly: <boolean>,
      buildIndexes: <boolean>,
      hidden: <boolean>,
      priority: <number>,
      tags: <document>,
      slaveDelay: <int>,
      votes: <number>
    },
    ...
  ],
  settings: {
    getLastErrorDefaults : <document>,
    chainingAllowed : <boolean>,
    getLastErrorModes : <document>,
    heartbeatTimeoutSecs: <int>
  }
}
```

Reconfigurar un replica set:

```
rs.reconfig(<config>)
```

6.5 Configuración rápida.

Existe una forma de crear un replica set de forma rápida para pruebas y desarrollo, para ello necesitamos el directorio `/data/db`

```
# mkdir -p /data/db
# chown -R student:student /data
$ mongo --nodb
> var rsTest = new ReplSetTest({name: "replicaTest", nodes: 3})
> rsTest.startSet() && rsTest.initiate()
```

Note: Una vez salgamos de la consola el réplica set se finaliza

Conectarse desde consola:

```
$ mongo --nodb
> conn = new Mongo("localhost:31000") // Conectamos al primario
> db = conn.getDB("test")
// Para conectarse al secundario
replicaTest:PRIMARY> conn = new Mongo("localhost:31001")
replicaTest:PRIMARY> db = conn.getDB("test")

// Volvemos al primario para insertar datos
> replicaTest:PRIMARY> use test // Esta puede no ser necesaria
```

CHAPTER

SEVEN

SHARDING

Para alto rendimiento deberíamos tener todos los índices cargados en memoria junto con el working set [1].

La forma tradicional ante llenados de memoria es escabilidad vertical (mas memoria o disco). Mongo con los shards nos permite escalabilidad horizontal.

Funcionamiento:

- Para particionar una colección usamos una "shard key"
- Para que una colección se particione automáticamente hay que activar sharding a nivel de base de datos y a nivel de colección.
- Las colecciones que no tengan shard se crean en el shard1 siempre (primary shard).
  - Con `sh.movePrimary(...)` podemos cambiar el shard que será el primario (moverá las colecciones)
- Según la shard key, inserta en la partición que corresponda entre "Min value" y "Max value"
- Una partición es un "chunk" no un "shard".
- Cuando una colección es pequeña inicialmente tiene una única partición (chunk) con un tamaño máximo por defecto de 64MB. (-inf .. inf). El chunch es en base a la shard key por lo que solo existe cuando estamos en un shard y la colección está marcada para sharding.
- Al llenarse MongoDB la divide en 2 chunks automáticamente (-inf.. 1000),[1000... inf) si el tamaño era de 2000 documentos todos del mismo tamaño. (realmente divide en 2 chunks de 32MB).
- En la división de chunks se pasa uno de los nuevos chunk al otro shard (parece que no tiene por qué ser siempre, sino que es mongos quien va balanceando los chunks entre shards)

7.1 shard key

- Una shardkey puede tener duplicados, se puede definir un nombre por ejemplo para que las escrituras vayan a varios shards
  - Por la misma razón, el objectId no es bueno para la shardkey porque los timestamp del principio obligaría a escribir todos los últimos en el mismo shard. (incrementales y timestamps no son buena idea)
- Una shardkey solo puede estar en un chunk, es decir, si es un valor muy usado, puede haber chunks con mas de 64MB (por ejemplo datos de los usuarios, con shardkey "santiago") -> jumbochunk.
- Para poder crear una shard key **debe existir un índice sobre el campo o un índice compuesto que empiece por ese campo**.
- Es inmutable
- Los valores son inmutables

[1] **Working set**. Datos que mas utilizamos y que nos interesa que estén en memoria directamente.

- Limitada a 512 bytes de tamaño *¿¿Comprobar??*.
- Es usada para distribuir las queries en un shard.
  - Por lo que se debería elegir un campo que sea usado comunmente en las queries
- can be unique across shards (puede, que no debe) *¿¿Comprobar??*.
  - '_id' field is only unique on individual shards

Consideraciones sobre la shard key. Reglas recomendadas a seguir, pero no son obligatorias ni tienen por que ser la solución óptima:

- Cardinality (alta cardinalidad). Veremos que nos pueden ayudar los índices compuestos.
- Write Distributions
- Query Isolation (que una query vaya sobre un shard siempre que sea posible)
- Reliability
- Index Locality

Important: *Cardinalidad y que no exista Hotspotting* (por ejemplo fecha actual, que siempre carga el mismo shard)

Estrategias que se suelen usar:

- basado en localización (IPs pueden estar incluidas aquí)
- incremental + (compuesto)
- **Bucketing**. Dependiendo de lo que queramos hacer, hay que potenciar unos elementos u otros.
- **Hash**.

Preguntas a hacer para elegir la clave:

- Crecimiento shard. Cómo esperamos que crecerá nuestro shard (3 nodos shard, o 100 nodos... mejor 3)
- Latencia que sufriremos (zonas geográficas separadas)
- CPU & RAM (tiempo de proceso real). Cómo va a crecer el trabajo
- Ratio de lecturas/escrituras sobre la colección

7.2 mongos

Demonio intermediario al que se conecta el cliente y decide porque shard pasa la query (hace de router).

Para saber a que shard manda la query usa los **config server** (tienen información de todos los chunks y donde están esos chunk).

Sabe a donde mandar la query cuando se usa en la query la shardkey sino hay que consultarlos todos.

En la práctica, se debería poner un mongos en cada máquina cliente.

7.3 config server

El config server debería ser una máquina dedicada (aunque tiene muy poca configuración). Podemos tener 1 o 3, no 2, (los 3 tienen los mismos datos), pero si se cae uno se convierten en solo lectura (de metadatos: no se puede cambiar la definición de los chunks, pero si se pueden realizar escrituras en colecciones)

Para activarlo hay que definir varios shard (puede ser un standalone o un replica set, mejor este último para tener tolerancia a fallos).

El mongos bloquea los config server durante la migración o split de chunks para que otro mongos no pueda escribir.

7.4 Configuración.

config server:

- Desde linea de comandos: `mongod ... --configsvr`
- Desde ficheros de configuración
  - `configsvr = true` (antiguo formato)
  - `sharding.clusterRole: configsvr` (YAML)

El resto de las configuraciones vistas para mongod son también válidas.

mongos:

Desde linea de comandos: `mongos --configdb <hostname and port of config servers>` (por defecto: 27017)

```
mongos ... --configdb mongodbvm:28001 // configuración con 1 config server
mongos ... --configdb mongodbvm:28001,mongodbvm:28002,mongodbvm:28003 // configuracón con 3 confi
```

shards:

Cada uno de los shard estará compuesto de un servidor mongod standalone o de un replica set (mejor este caso para tolerancia a fallos)

Cada uno de los `mongod` deberán tener la opción `shardsvr` activa para indicar que forman parte de un shard cluster.

- Desde linea de comandos: `mongod ... --shardsvr`
- Desde ficheros de configuración
  - `shardsvr = true` (antiguo formato)
  - `sharding.clusterRole: shardsvr` (YAML)

Inicialización:

Una vez están los demonios ejecutándose, nos conectamos con el comando `mongo` al demonio `mongos`

Una vez conectado, para añadir un shard al cluster:

```
sh.addShard("localhost:27001") // shard con un mongod standalone.
sh.addShard("rep0/localhost:27001") // shard con un replica set.
```

Note: En el caso de replica sets, sólo es necesario añadir un nodo. El sistema ya obtiene toda la configuración del shard.

Ahora necesitamos habilitar el sharding para la base de datos y para las colecciones que queramos:

```
sh.enableSharding("dbname")
//Field es el campo que será la shard key.
sh.shardCollection("dbname.collectionname",{field:1})
```

7.5 Configuración rápida.

Al igual que los replica sets, se puede crear un shard rápido de prueba

```
$ mongo --nodb
> shard = new ShardingTest({name: "testShard", shards: 3, chunksize: 26,
                rs: { dbpath:"/data/shard"}})
```

Info, para mover colecciones entre tags:

```
mongos> db.runCommand({movePrimary: "tweets", to: "shard0000"})

mongos> db.tweets.ensureIndex({"user.screen_name": 1})
//No lo crea automáticamente el shard key, con lo que es bueno hacerlo antes.
mongos> sh.enableSharding("twitter")
mongos> sh.shardCollection("twitter.tweets", {"user.screen_name":1})
```

CHAPTER

EIGHT

BACKUPS Y RESTAURACIÓN

DR (Disaster recovery) vs HA (High availability)

Depende de las necesidades del negocio, ¿podemos perder datos de un día? ¿cuanto tiempo podemos estar offline?...

Si perdemos un nodo, al conectar un nodo limpio se replican los datos desde el primario, pero esto significa una carga de la red al transpasar datos. Por lo que puede ser interesante cargar de un dump (o por si perdemos todo el replicaSet).

8.1 Backups lógico: mongodump/mongorestore.

2 opciones:

- Conectándose a mongod
- Leyendo datos del dbpath directamente

Genera para cada colección 2 ficheros:

- documentos en formato BSON
- metadatos en formato JSON (estos metadatos son basicamente la definición de los índices)

Con el comando `bsondump` podemos ver los datos existentes en los ficheros bson.

Ejemplos:

- Para crear un dump completo de todas las BD del replica set (menos BD "local")

```
$ mongodump --port 27002
```

- Dump de una sola base de datos indicando en que directorio se generará (por defecto se genera en "dump")

```
$ mongodump --port 27002 -d twitter --out backup/misbackups/twitter
```

Para restaurar hay que conectarse al primario o sacar el secundario fuera del replicaset para cargar los datos

```
$ mongorestore -d twitter --port 27001 backup/misbackups/twitter/twitter
```

Note: Si se restaura sobre una BD existente, crea los documentos nuevos si no existen (por clave primaria) o da un error (que no se muestra en consola)

Este backup puede ser inconsistente, ya que pueden estar modificándose datos mientras se está haciendo el backup. Es decir, no bloquea la BD. Para tener un backup consistente hay 2 opciones:

- Bloquear: `db.fsyncLock()/db.fsyncUnlock()` (Esta es una buena opción para no tener que parar el servidor) o parar el servidor.
- `mongodump` con `--oplog`. Guardo las bases de datos y las operaciones ejecutadas durante el dump de la base de datos.

El fsyncLock además de bloquear hace una sincronización de memoria con disco, para que los ficheros de datos tengan los datos actualizados.

Mongodump es el mecanisco de backup mas lento. El backup físico es mas rápido, por lo que ante servidores standalone, ya que hay que bloquear para hacer backups existentes, es mejor un backup físico.

Para la opción –oplog tiene que ser un backup completo.

```
$ mongodump --port 27002 --out backup/misbackups/withOpLog --oplog
```

Esto creará además de los archivos bson y json anteriores, un nuevo fichero oplog.bson

Para restaurar un dump con oplog:

```
$ mongorestore --port XXX --oplogReplay dumpFolder
```

Se podrían simular backups incrementales haciendo backups del oplog (no existe una manera nativa de backups incrementales)

```
$ mongodump --port 27002 -d local -c oplog.rs --out outfolder
$ mv oplog.rs.bson oplog.bson
$ mongorestore --port XXX --oplogReplay ...
```

Para filtros selectivos en el dump tenemos "`--query/filter`"

Si queremos restaurar en una base de datos con distinto nombre simplemente:

```
$ mongorestore --port XXX -d nuevoNombre pathToDump
```

8.2 Backups físicos

Esta es una copia de ficheros. Y para que sean consistentes, tenemos varias opciones:

- shutdown / fsyncLock() ó...
- snapshot del sistema de ficheros y luego copia de estos ficheros. Minimiza los tiempos de backup con lo que son consistentes.

  Important: Importante tener el journaling activado (para poder hacerlo sin apagar el servidor) y que el journal esté en el mismo sistema de ficheros.

  Si el journal está en otro volumen, primero deberíamos hacer un bloqueo, hacer snapshots de todos los filesystems y quito el bloqueo. Luego se puede hacer la copia de ficheros a partir del snapshots.

Con la copia física hay que copiar una BD completa siempre. No es posible copiar una sola colección.

8.2.1 En un shard

1. Hacer backup de cada uno de los replica set y de los config server.
2. Paramos balanceador
3. Paramos un config server (para que no se modifiquen los metadatos de los chunk)
4. Backup de cada shard y de un config server
5. Reiniciamos config server
6. Reiniciamos balanceador

Info: La gente de MongoDB tiene MMS Backup. De esta forma la gente de Mongo es quien hace el backup.

CHAPTER

NINE

SEGURIDAD

MongoDB tiene la autenticación/autorización **desactivada por defecto**.

Esto es debido a que la primara opción de seguridad debería ser **establecer la red de cluster en una red privada protegida por cortafuegos**.

Cifrado

La comunicación entre los distintos nodos y cliente no está cifrada, ni dispone de cifrado alternativo. Se podría usar aplicaciones de terceros (tunel ssh, ...). Hay que tener en cuenta que el cifrado puede penalizar el rendimiento.

Tampoco hay cifrado de datos en disco.

Lo único que hay cifrado es la contraseña de los usuarios (tanto en el servidor como en la red).

Autenticación:

Tenemos 2 tipos de autenticación:

Para un **cliente**, especificando usuario y contraseña. Asociada a esta autenticación tenemos privilegios de usuarios.

Los usuarios se crean a nivel de base de datos.

- `use twitter; db.addUser("...")`. También `db.createUser(...)` desde la 2.6. Con este último comando podemos crear un usuario para varias BD.
- Se guardan en la colección `system.users` en cada una de las BD.
- Debemos activar la autenticación con `--auth` en el arranque del demonio.
- Desde localhost se permite conectar sin autenticación si no hay usuarios definidos en `admin.system.users`.

Para usar **autenticación con un shard o réplica set**, primero debemos generar una clave:

```
$ openssl rand -base64 741 > mongodb-keyfile
```

Y cambiar los permisos

```
$ chmod 600 mongodb-keyfile
```

Lo que nos queda es arrancar los replicaSet y shard con la opción `--keyFile` (implica `--auth`)

La clave es para evitar suplantaciones de identidad de un réplica set, lo cual sería posible si se cae un elemento del réplica set y se suplanta la IP.

Ejemplo configuración con clave

```
# Config servers
mongod --dbpath=data/config_server_1 --fork \
     --logpath=data/config_server_1/mongod.log --logappend \
     --port 28001 --configsvr --keyFile data/mongodb-keyfile
mongod --dbpath=data/config_server_2 --fork \
     --logpath=data/config_server_2/mongod.log --logappend \
     --port 28002 --configsvr --keyFile data/mongodb-keyfile
mongod --dbpath=data/config_server_3 --fork \
     --logpath=data/config_server_3/mongod.log --logappend \
     --port 28003 --configsvr --keyFile data/mongodb-keyfile

# RS set0
mongod --dbpath=data/rs1/db1 --journal --fork \
     --logpath=data/rs1/db1/mongod.log --logappend --replSet set0 --port 27001 \
     --shardsvr --keyFile data/mongodb-keyfile
mongod --dbpath=data/rs1/db2 --journal --fork \
     --logpath=data/rs1/db2/mongod.log --logappend --replSet set0 --port 27002 \
     --shardsvr --keyFile data/mongodb-keyfile
mongod --dbpath=data/rs1/db3 --journal --fork \
     --logpath=data/rs1/db3/mongod.log --logappend --replSet set0 --port 27003 \
     --shardsvr --keyFile data/mongodb-keyfile

# RS set1
mongod --dbpath=data/rs2/db1 --journal --fork \
     --logpath=data/rs2/db1/mongod.log --logappend --replSet set1 --port 27101 \
     --shardsvr --keyFile data/mongodb-keyfile
mongod --dbpath=data/rs2/db2 --journal --fork \
     --logpath=data/rs2/db2/mongod.log --logappend --replSet set1 --port 27102 \
     --shardsvr --keyFile data/mongodb-keyfile
mongod --dbpath=data/rs2/db3 --journal --fork \
     --logpath=data/rs2/db3/mongod.log --logappend --replSet set1 --port 27103 \
     --shardsvr --keyFile data/mongodb-keyfile

# MongoS
mongos --configdb mongodbvm:28001,mongodbvm:28002,mongodbvm:28003 \
     --logpath=data/mongos1/mongos.log --logappend --fork --port 28000 \
     --keyFile data/mongodb-keyfile
```

CHAPTER

TEN

MONITORIZACIÓN Y RENDIMIENTO

mongostat

- opcounters
- resident memory size
- page faults (cuando no lo encuentra en memoria y pasa a buscarlo a disco) Tener pocos es normal, tener muchos no es bueno, puede indicar que el working set no coje en memoria.
- Porcentaje de bloqueos y número de operaciones esperando en colas de escritura/lectura (ar/qr)
- (Mirar en la documentación de mongo online para ver una descripción de todos los campos)

mongotop

db.currentOp(true). Me enseña las operaciones actuales.

db.killOp(op_id). Para matar una operación que no está bloqueando o tardando demasiado.

db.serverStatus(). Estado general del servidor, información del flush (veces, tiempo que tarda de media, tiempo del último), número de conexiones, información del journal

db.stats()

db.<collection>.stats()

Database profiling

`db.setProfilingLevel()`. Permite establecer estadísticas de las queries que está ejecutando. Por defecto tiene level 0 (no está activado). level 1 -> queries lentas. level 2 -> todas las queries

Por defecto una query se considera lenta si tarda mas de 100ms.

`db.setProfilingLevel(2, -1)`. Con -1 para que muestre todas las queries, 0 puede dejar queries sin mostrar por los redondeos.

Esto crea una colleción llamada `system.profile` en la base de datos local (no se soporta sobre un sharding)

Tendrá una sobrecarga de acceso a disco, por lo que no es normal activarlo siempre, o como mucho las queries lentas. Dejando activado todo solo en temas de log

CHAPTER

ELEVEN

OPTIMIZACIÓN

- Se pretende optimizar las queries de lectura para alto rendimiento aunque se penalicen las escrituras.
- MongoDB trabaja con estructuras B-Tree para los índices.
- En la parte de sharding, no se pueden modificar los índices. *Duda: ¿solo shard key o todos?*
- Límite de índices por colección: 64
- La ordenación en memoria sin un índice está limitada a 32MB
- En una query solo se puede utilizar 1 índice, salvo en una query con operador $or.
  - En el caso del $or se puede usar un índice por cada una de las clausulas alternativas.
- Aunque eliminemos una colección, el índice permacene (está asociado al espacio de nombres: fichero .ns)

11.1 Explain

Lo ejecutamos para la query:

```
db.users.find().explain()
```

Cuando no existe un índice nos devuelve un `BasicCursor`

```
> db.users.find({age: {$gt: 80}}).explain()
{ "cursor" : "BasicCursor",
  "isMultiKey" : false,
  "n" : 19101,
  "nscannedObjects" : 100000,
  "nscanned" : 100000,
  "nscannedObjectsAllPlans" : 100000,
  "nscannedAllPlans" : 100000,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 781,
  "nChunkSkips" : 0,
  "millis" : 92,
  "server" : "mongodbvm:27017",
  "filterSet" : false
}
```

Tip:

- **"n" : 19101**. Número de documentos que coinciden con el criterio
- **"nscanned" : 100000**. Índices y/o documentos leídos
- **"millis" : 92**.
- **"scanAndOrder"**. Si está a "true" indica que estamos utilizando un índice pero que **NO** nos sirve para ordenar.

11.2 Tipos de indexaciones

- Simple (un solo campo)
- Compuesta (varios campos)
- MultiKey (arrays, subdocumentos o arrays de subdocumentos)
- TTL
- Text
- Geo (geoespaciales)
- Hashed Index

Comandos:

```
//Crear un índice
> db.users.ensureIndex({age: 1})
//Con unicidad
> db.users.ensureIndex({name:1},{unique:1})
//Elimina los duplicados si existen (dejando el primero)
> db.users.ensureIndex({name:1},{unique:1, dropDups : 1})
//Índice disperso (puede o no existir el atributo)
> db.nums.ensureIndex({z:1},{unique:true, sparse:true})
//Construye el índice en background
> db.nums.ensureIndex({z:1},{background:true})
//Índice TTL
> db.logs.ensureIndex({creado : 1}, {expireAfterSeconds: 5})
//Índice de tipo texto
> db.tweets.ensureIndex({text: "text"})
//Índice geoespacial (primera versión)
> db.tweets.ensureIndex({attrib: "2d"})
//Índice geoespacial (última versión)
> db.tweets.ensureIndex({attrib: "2dsphere"})
//Índice por hash
> db.collection.ensudeIndex( {_id: "hashed"})
//Para ver los índices existentes
> db.users.getIndexes()
//Para borrar un índice
> db.users.dropIndex("city_1")
//Borra todos los índices de una colección
> db.tweets.dropIndexes()
```

Cuidado con los índices dispersos y `$exists`, ya que puede evitar el uso del índice u otros problemas...

Se podría insertar un z=null que sería un valor a tener en cuenta, pero no se permitiría otra tupla con z=null (por el índice único sobre z)

Durante la construcción en background podrían existir escrituras en medio, y será mas lento. A partir de la versión 2.6 se puede construir mas de un índice en background al mismo tiempo.

11.3 Índice Simple.

- Indicando campo y orden
- Se crea un índice con punteros fuertes a cada documento

```
> db.users.ensureIndex({age: 1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}

> db.users.find({age: {$gt: 80}}).explain()
{
    "cursor" : "BtreeCursor age_1",
    "isMultiKey" : false,
    "n" : 19101,
    "nscannedObjects" : 19101,
    "nscanned" : 19101,
    "nscannedObjectsAllPlans" : 19101,
    "nscannedAllPlans" : 19101,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 149,
    "nChunkSkips" : 0,
    "millis" : 81,
    "indexBounds" : {
        "age" : [
            [
                80,
                Infinity
            ]
        ]
    },
    "server" : "mongodbvm:27017",
    "filterSet" : false
}


> db.users.ensureIndex({city: 1})

> db.users.find({city: /^B/i}).explain()
{
    "cursor" : "BtreeCursor city_1",
    "isMultiKey" : false,
    "n" : 22210,
    "nscannedObjects" : 22210,
    "nscanned" : 100000,
    "nscannedObjectsAllPlans" : 22210,
    "nscannedAllPlans" : 100000,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 781,
    "nChunkSkips" : 0,
    "millis" : 230,
    "indexBounds" : {
        "city" : [
            [
                "",
                {

                }
            ],
            [
                /^B/i,
                /^B/i
            ]
```

```
        ]
    },
    "server" : "mongodbvm:27017",
    "filterSet" : false
}
```

Tip: El índice no funciona porque en este caso se está usando una expresión regular.

Ante varios índices el planificador usa el índice que cree mas conveniente:

```
> db.users.find({city:"Barcelona",age: {$gt: 80}}).explain()
{
    "cursor" : "BtreeCursor city_1",
    "isMultiKey" : false,
    "n" : 2137,
    "nscannedObjects" : 11192,
    "nscanned" : 11192,
    "nscannedObjectsAllPlans" : 11704,
    "nscannedAllPlans" : 12220,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 95,
    "nChunkSkips" : 0,
    "millis" : 48,
    "indexBounds" : {
        "city" : [
            [
                "Barcelona",
                "Barcelona"
            ]
        ]
    },
    "server" : "mongodbvm:27017",
    "filterSet" : false
}
```

11.4 Índice Compuesto

```
> db.users.ensureIndex({city: 1, age: 1})
```

Si se busca por un campo del índice se puede seguir usando un índice compuesto siempre que este sea el primer nivel del índice (el orden importa a la hora de crear los índices compuestos).

Note: Si se recupera solo la información que está en el índice, es una query muy rápida y se denomina **"Query Covered"**. Para esto los campos de la búsqueda y de la proyección deben estar en el índice:

```
> db.users.find({city:"Barcelona"}, {age:1, _id:0}).explain()
```

Podemos añadir mas niveles al índice

```
> db.users.ensureIndex({city: 1, age: 1, name:1}, {name:"byCityAgeAndName"}) //Con nombre

> db.users.find({city: "Barcelona", name : /9$/}).explain()
{
    "cursor" : "BtreeCursor city_1_age_1_name_1",
    "isMultiKey" : false,
    "n" : 1098,
    "nscannedObjects" : 1098,
    "nscanned" : 11192,
```

```
    "nscannedObjectsAllPlans" : 3150,
    "nscannedAllPlans" : 13246,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 103,
    "nChunkSkips" : 0,
    "millis" : 77,
    "indexBounds" : {
        "city" : [
            [
                "Barcelona",
                "Barcelona"
            ]
        ],
        "age" : [
            [
                {
                    "$minElement" : 1
                },
                {
                    "$maxElement" : 1
                }
            ]
        ],
        "name" : [
            [
                "",
                {

                }
            ],
            [
                /9$/,
                /9$/
            ]
        ]
    },
    "server" : "mongodbvm:27017",
    "filterSet" : false
}
```

11.5 Índice Multikey

Cuando se hace sobre un atributo que es un array, un subdocumento o arrays de subdocumentos.

Se pueden hacer índices multikey combinandolas con “simple”, pero es mejor que el primer nivel sea el simple

Sobre atributos de un subdocumento (no lo toma como un multiKey, pero se categoriza aquí):

```
> db.tweets.ensureIndex({"user.followers_count" : 1, "user.friends_count" : 1})

> db.tweets.find({"user.followers_count" : {$gt : 1000}}).explain()
{
    "cursor" : "BtreeCursor user.followers_count_1_user.frinds_count_1",
    "isMultiKey" : false,  <----- multikey
    "n" : 5862,
    ...
```

11.6 Índices TTL

Estos son los índices con tiempo de vida, como por ejemplo logs rotativos, o almacenamiento de sesiones

Existen 3 restricciones para este tipo de índices:

- El campo a indexar tiene que ser un ISODate (obligatorio).
- Tiene que estar en un campo limpio, sin que esté usado en un prefijo de un índice (sin que se use en otro índice).
- No puede ser una clave compuesta.

Ej.:

```
> use logger
//No se puede garantizar que sea justo a los 5s
> db.logs.ensureIndex({creado : 1}, {expireAfterSeconds: 5})
> db.logs.insert({creado: new Date(),
        msg: "Prueba 001", detalles: "Prueba 001", nivel_log: "FINE"})
> db.logs.find() // después de 5s desaparece el documento
```

Otra alternativa, expire en 0 y añadiendo la fecha de expiración en el insert:

```
> db.subastas.ensureIndex({fecha_final:1},{expireAfterSeconds: 0})
> db.subastas.insert({id:90, product: {id: 4, precio: 56.8}, pujas:[],
        fecha_final: new Date("20:15:00")})
```

11.7 Índices TEXT

Indice sobre textos complejos.

Como son lentos, es normal montarlos en background. Cuando trabajemos sobre índices TEXT siempre trabajaremos con accesos a disco.

A la hora de definir un índice, en vez de un "1", estableceremos un "text".

```
> db.tweets.ensureIndex({text: "text"})
```

Para hacer búsquedas sobre texto será el operador $text (tanto en CRUD como en el framework de agregación).

```
> db.tweets.find({$text: {$search: "preciso"}}).explain()
```

Note: Siempre es un OR, no existe un Y.

Restricción: Solo puede existir un único índice de tipo "texto".

Se puede escoger un idioma o "none" (por defecto es el del sistema):

```
> db.tweets.ensureIndex({text: "text"},{default_language: "fr"})
> db.tweets.ensureIndex({text: "text"},{default_language: "none"})

// Busca con OR
> db.tweets.find({$text: {$search: "depende de la velocidad"}})

// Busqueda por frase
> db.tweets.find({$text: {$search: "\"depende de la velocidad\""}})

// Busqueda por frase AND otro término (Entre frases y palabras AND, entre palabras OR)
> db.tweets.find({$text: {$search: "\"depende de la velocidad\" cosaDistinta"}})

//Especificando idioma
```

```
> db.tweets.find({$text: {$search: "\"depende de la velocidad\" cosa distinta",
        $language: "es"}})
```

11.8 Índices geoespaciales

Tipo "2d".

- Plano
- Los primeros en aparecer en Mongo.
- Las coordenadas son sobre 100,100 (si queremos mas valores hay que especificar una opción)
- La estructura a mantener es un array simple donde añadiremos la longitud y la latitudes ([long, latitud])
- Se utiliza como entorno legacy

Tipo "2dsphere".

- El segundo que se añadió y el que se debería utilizar en la mayoría de los casos
- Sigue la especificación de GeoJSON
- http://docs.mongodb.org/manual/tutorial/build-a-2dsphere-index/
- La localización es un subdocumento donde se define el tipo geométrico que vamos a tener ("Point", "Line", "Point[]", "Polygon", * "Circle")
  - `{loc: {type : "Point", coordinates: [long, lat]}}`

Important: Con "2dsphere" las unidades son en metros, mientras que con "2d" son en radianes.

```
> db.<collection>.ensureIndex({<location field>:"2d", <additional field>: <value>}, ...)

> db.<collection>.find( { <location field> :
        { $geoWithin: {$box|$polygon|$center : <coordinates>}}})
```

Se puede usar $near (en los 2dsphere solo??) como alternativa a $geoWithin (cerca vs dentro de una figura)

Con $near sale un listado de documentos ordenados por cercanía

```
{
  $near: {
     $geometry: {
        type: "Point" ,
        coordinates: [ <longitude> , <latitude> ]
     },
     $maxDistance: <distance in meters>,
     $minDistance: <distance in meters>
  }
}

> db.tweets.find({coordinates : {$near: {$geometry :
        {type : "Point", coordinates : [107.119031, -6.301964] }}}}).count()
> db.tweets.find({coordinates : {$near: {$geometry :
        {type : "Point", coordinates : [107.119031, -6.301964] },
        $maxDistance : 20000}}}).count() //20km de distincia máxima
```

11.9 Índices Hash

Desde la versión 2.4 de MongoDB

Podemos utilizar el campo que queramos y marcarlo dentro del index como “hashed”.

En vez de trabajar con un documento de texto grande, se trabajaría con un hash para realizar la indexación. Es una buena opción para elegir como clave de shard key en determinadas colecciones.

```
> db.collection.ensudeIndex( {_id: "hashed"})
```

CHAPTER

TWELVE

DESARROLLO. INTRODUCCIÓN.

Tanto el diseño como a la hora de trabajar, mongodb es muy dependiente de la aplicación.

Si no se pone "_id" en el insert se genera ObjectId que está compuesto de: timestamp+idMachine+pid+sequencia => byte

JSON admite de tipos de datos:

- double
- string
- array
- sub-docs
- boolean
- null

Para tener otros tipos de datos usamos BSON, que es mucho mas rico.

Todo lo que hagamos desde la consola, aunque sea JSON se serealiza en BSON (http://bsonspec.org)

Como "_id" también se puede usar un documento:

```
{_id: {state: " ", city: " "}}
```

Hay que tener en cuenta que en estos casos influyen campos, orden y valor como clave primaria ya que lo que se tienen en cuenta como unicidad es el hash.

Esto está permitido, siendo 2 documentos distintos:

```
> db.lines.insert({_id: {number: 01, label:"002"} , origen:"or2", destino:"destino2"})
> db.lines.insert({_id: {label:"002", number:01} , origen:"or2", destino:"destino2"})
```

Se pueden insertar arrays directamente que es mas eficiente que insertar uno por uno

```
> var array = [
... {number:1, label:"001"},
... {number:2, label:"002"}
... ]
> db.lines.insert(array)
BulkWriteResult({
"writeErrors" : [ ],
"writeConcernErrors" : [ ],
"nInserted" : 2,
"nUpserted" : 0,
"nMatched" : 0,
"nModified" : 0,
"nRemoved" : 0,
"upserted" : [ ]
})
```

En una lista, si hay un error en las operaciones, como la transaccionalidad es a nivel de documento, pueden quedar una parte de ellos insertada y la otra no.

CHAPTER

THIRTEEN

DESARROLLO. DRIVER

Podemos descarganos el driver según el lenguage necesario en: http://docs.mongodb.org/ecosystem/contents

MongoClient -> Mongo (driver)

La app se comunicará con ayuda del driver al mongod (primario o secundario) o al mongoS.

MongoClient nos ayuda ya con problemas de conexion (pool) o concurrencia. Nos abre un pool de conexiones.

El objeto `DBObject` es usado para manejar JSON, pero es abstracto. `BasicDBObject` será la instancia normal a usar.

Ejemplos trabajando con groovy

```
import com.mongodb.*;
import com.mongodb.util.*;

class TestMongoGroovyClient {

    static void main(def args) {
        MongoClient client = new MongoClient("localhost", 27017)

        println client.getDatabaseNames() //show dbs

        DB twitterDB = client.getDB("tweeter")

        println twitterDB.getCollectionNames() // show collections

        DBCollection tweets = twitterDB.getCollection("tweets")

        BasicDBObject firstTweet = tweets.findOne()

        // Buscar los usuarios que son de Santiago de compostela
        DBObject query = new BasicDBObject(
                "user.location", "Santiago de Compostela")

        tweets.find(query).each { println it.text }

        // Todos aquellos usuarios que tienen mas de 100 followers
        def query2 = new BasicDBObject("user.followers_count",
                new BasicDBObject('$gt', 100))
        println tweets.count(query2)

        def query3 = query2.append("user.location", "London")
        println tweets.count(query3)

        String queryStr = "{'user.location': 'London',
                'user.followers_count': {\$gt: 100}}"
        BasicDBObject json = (BasicDBObject) JSON.parse(queryStr)
        println tweets.count(json)

        QueryBuilder query4 = QueryBuilder.start(
```

```
                "user.followers_count").greaterThan(100)
        println tweets.count(query4.get())

        //proyecciones
        tweets.find(query4.get(), new BasicDBObject("text", 1)).limit(10).each {
            println it
        }

        // BasicDBList para manejar arrays

    }
}

import java.nio.channels.ReadPendingException;

import com.mongodb.*;
import com.mongodb.util.*;

class TestWithReplicaSet {

    static void main(def args) {
        // No hay que indicar cual es el primario ni nada
        MongoClient client = new MongoClient([
            new ServerAddress("localhost", 31000),
            new ServerAddress("localhost", 31001),
            new ServerAddress("localhost", 31002),
            ])

        println client.getDatabaseNames()

        DB db = client.getDB 'test'
        DBCollection users = db.getCollection 'user'

        users.find().each { println it }

        // El insert básico no garantiza que se lanza una excepción si falla
        // hay que modificar el write concern
        //users.insert(new BasicDBObject("username", "juan"))


        /*
         * Para modificar las preferencias de lectura
         * ReadPreference.primary() // Solo de primario
         * ReadPreference.primaryPreferred() // Primario preferido,
         *                                      pero podemos leer de secundarios
         * ReadPreference.secondary() // Solo de secundarios
         * ReadPreference.secondaryPreferred() // Secundario preferido
         *
         * Por defecto es primaryPreferred
         */

        //Si el primario se cae no quiero leer de ninguno
        users.setReadPreference ReadPreference.primary()

        /*
         * El writeConcern se puede hacer a nivel de cliente,
         * colección, insert, update...
         */
        // Devuelve el acuse de recibo cuando replique en la mayoría
        users.setWriteConcern(WriteConcern.MAJORITY)
        // Definir a cuantos hay que escribir. W=0 (0 es el valor por defecto)
        users.setWriteConcern(WriteConcern.NORMAL) //W=0
        //W=1 (inserción en el primario pero sin llegar al journal)
```

```
        //WriteConcern.SAFE
        //WriteConcern.ACKNOWLEDGED //también W=1

        // Si queremos personalizar el WriteConcern
        try {
            // 4 nodos, 2000ms timeout, <?>, esperamos al journal = true (W=4|j=1)
            users.insert(new BasicDBObject("username", "juan"),
                    new WriteConcern(4, 2000, true, true))
        } catch (WriteConcernException e) {
            e.printStackTrace()
        }

        try {
            users.insert(new BasicDBObject("username", "juan"),
                    new WriteConcern(4, 2000, true, true))
        } catch (e) {
            e.printStackTrace()
        }

    }
}
```

CHAPTER

FOURTEEN

DESARROLLO. DISEÑO

Diseño normalizado (favorecemos escrituras) vs Diseño no normalizado (favorecemos lecturas pero hay duplicados).

A la hora de diseñar con mongo se suele tirar hacia el lado del diseño no normalizado.

Note: Hay que intentar siempre minimizar la lista de documentos que se devuelven al driver

---

Ejemplo:

Queremos almacenar un conteo de los accesos realizados sobre un recurso. Hay que tenerlos controlados por día, hora y minuto

Consultas típicas: información de conteos, recursos mas vistos...

Elegir posibles candidatos a shardkey.

- sample: ip, http_method, http_code, fecha, ...
- recurso: /index.htm, /productos...
- hit -> contador

Una opción sería:

```
{_id, ip, http_method, http_code, fecha, fecha_completa, recurso}
shardKey = recurso, fecha

Con una agregación realizar crear la nueva colección que se usará en las consultas típicas
{ id={dia, hora, minuto}, hits}
```

Ejemplo:

Dados los datos de los empleados: información básica: nif, nombre, ...

Queremos obtener informes de gastos por empleado, mes y año, de la forma:

- gasto
  - proyecto
    - tiempo
    - concepto
    - importe
  - total
  - estado (nuevo, emitido, confirmado, pagado, denegado)

Para confirmado, pagado o denegado hay una serie de empleados (técnicos o lo que sea) que pueden modificar el estado del gasto

```
empleados:
{
  id:
  nombre_empleado:
  inf_basica:
  supervisa: [<ids empleados>]
}

gastos:
{
  _id: {proyecto: "proyecto", anho:..., mes:...}
  gastos : [
    {
      _id:
      id_empleado:
      tiempo:
      concepto:
      gasto:
      estado:
    }
  ]
}
```

CHAPTER

FIFTEEN

AGGREGATION FRAMEWORK

Existen 3 formas:

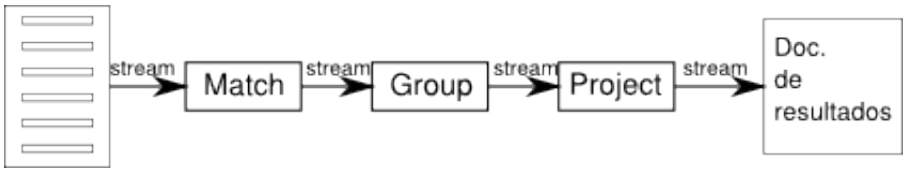
- Usando Count, group, ...
- Usando MapReduce
- Usando "aggregation". La forma por la que vamos a empezar, que es similar a lo posible en SQL
  - `> db.coll.aggregation([ ... ])`

Los drivers tienen esta funcionalidad, pero lo que se hace es producir un JSON que se envía al motor.

Hay que tener cuidado con memoria, CPU, primarios de replica set y sobre todo mucho cuidado con el sharding

La forma de trabajar es como el "pipe" de los sistemas linux. Pasaremos la salida de un comando a otro y así sucesivamente...

Ejemplo: "nDocs =stream=> fase de match =stream=> group =stream=> project =stream=> documento final"



Para poder realizar operaciones en paralelo hay que usar "Map Reduce", el resto son secuenciales.

15.1 Operadores

- **$match**. igual a queries (find({}))
- **$project**. como las proyecciones del find pero le podemos añadir mas elementos (sumas, divisiones...)
- **$group**. Parecido a db.coll.group() pero no es una función igual
- **$unwind**. Explotar el array -> Dado un array de 3 elementos, generamos 3 elementos individuales.
- **$sort**. Parecido a trabajar con el cursor, pero trabajamos con el stream.
- **$limit**.
- **$skip**.
- **$out**. Escritura en fichero

Limitaciones:

- **Pipeline operator memory limits: 100MB - No se puede cambiar**.
- Lo que podemos hacer es habilitar el uso en disco: `allowDiskUse: true`

15.1.1 $match.

- Filtra documentos como un find.
- No nos valen operaciones geoespaciales
- No es válido tampoco el $where

El resultado de la operación no es un cursor (no es un array). Es un documento que contiene los resultados.

```
> db.zips.aggregate([ {$match: {state : "AL" }}])

//Ejemplo encadenando. ``$city`` es el valor del campo
> db.zips.aggregate([ {$match: {state : "AL" }},
        {$project: {_id:0, city: {$toUpper: "$city" }}}])
```

15.1.2 $group

Plantilla para los siguientes documentos (por la agrupación)

Cuidado, ya que se procesa toda la información en memoria

Ejemplo: calcular la población total por estado:

```
> db.zips.aggregate({$group: {_id: "$state", pop_total: {$sum: "$pop"}}},
        {$sort : {pop_total : -1}})
> db.zips.aggregate({$group: {_id: "$state", pop_total: {$sum: "$pop"}}},
        {$sort : {pop_total : -1}}, {$limit: 2})
{ "_id" : "CA", "pop_total" : 29760021 }
{ "_id" : "NY", "pop_total" : 17990455 }
```

Ejemplo: media de población de los estados.

```
> db.zips.aggregate({$group: {_id: "$state", pop_total: {$sum: "$pop"}}},
        {$group: {_id: "total", avg: {$avg: "$pop_total"}}} )
{ "_id" : "total", "avg" : 4876664.176470588 }
```

Proyecciones encadenadas:

```
> db.zips.aggregate([ {$group: {_id: {state: "$state", city: "$city"},
        pop_total: {$sum: "$pop" }}},
        {$group: {_id: "$_id.state", pop_media : {$avg : "$pop_total"}}},
        {$sort : {pop_media : -1}}])
```

15.1.3 $project

Proyecta los campos realizando operaciones si es necesario

15.1.4 $unwind

Hace una explosión de los elementos de un array a elementos simples.

Si el campo no es un array da un error

15.1.5 $sort, $limit, $skip

Funcionan igual que en las queries normales.

Cuidado con el tema de memoria de nuevo.

15.1.6 $out

Sería siempre la última fase y crearía una nueva colección en la BD.

```
{$out: "media_x_states"}
```

15.2 Sharding

En el framework de agregación, si hacemos un match, el uso va a ser el mismo que sin sharding (se hace un match local, se juntan y se devuelven). Con la proyección pasa lo mismo.

El problema son en las agrupaciones que realizan cálculos o se necesitan todos los documentos (group, sort). Que tienen que traer todos los datos al primary shard para realizar todos las operaciones.

Podemos mejorar el rendimiento con el orden de operaciones en los shards, de forma que limitemos los resultados:

-> match, project, group, sort, project...

Si esto se nos queda corto (manejar una gran cantidad de operaciones para agregaciones), habría que plantearse otras soluciones alternativa a Mongo.

La alternativa mas usada es Hadoop (existe connector mongo-hadoop). Y en cantidades mayores: mahout, Hive/Pig, spark/storm, ...

Mongo -> Hadoop -> Mahout, Hive...

Podemos ver un explain en agregación con `{explain: true}`:

```
> db.tweets.aggregate([ {$match: {"user.friends_count": {$gt: 0}}},
        {$match: {"user.followers_count": {$gt: 0}}},
        {$project: { "user.screen_name": 1,
            "user.friends_count": 1,
            "user.followers_count": 1,
            _id: 0,
            ratio: {$divide:["$user.followers_count", "$user.friends_count"]}}
        },
        {$group:{_id: 1, avg: {$avg: "$ratio"}}}
    ], {explain: true} )
```

15.3 MapReduce

http://docs.mongodb.org/manual/tutorial/map-reduce/

Se usaba inicialmente, cuando no había funciones de agregación

Con map reduce hay que ir haciendo por fases, no hay pipeline:

Fase mapeo: filtro, parseo, limpieza, proyección, split de texto... Esto produce unos valores K,V (clave, valor)

Fase reducción: operaciones de agregación: sumas, medias, ...

```
map = function() {
    emit( this.user.name, { "diggs": this.diggs, "posts" : 1 } ) ;
}

reduce = function(key, values) {
    var diggs = 0;
    var posts = 0;
    values.forEach(function(doc) {
        diggs += doc.diggs;
        posts += doc.posts;
    } ) ;
```

```
    return { "diggs" : diggs, "posts" : posts } ;
}

db.stories.mapReduce( map, reduce, { "out" : "digg_users" } ) ;
```

Important:

1. Can be run on secondary nodes to keep from bogging down primary's performance.
2. Can only at most 16MB of result.

CHAPTER

SIXTEEN

CERTIFICACIÓN

Preguntas típicas de certificación:

- **WriteConcern y ReadPreferences sobre clientes o tablas**.
- **¿Cómo mejorar el rendimiento de las escrituras?**. Con un WriteConcern de W=0 o menos
- Sobre unas 10 preguntas van a ser sobre el framework de agregación.

CHAPTER

SEVENTEEN

ANEXO. OTROS RECURSOS

- MongoDB-Performance-Best-Practices.pdf
- Ejemplos encontrados de examenes:
  - http://blog.cloudthat.in/sample-questions-for-mongodb-certified-dba-c100dba-exam/
  - http://blog.cloudthat.in/sample-questions-for-mongodb-certified-dba-c100dba-exam-part-ii/

CHAPTER

EIGHTEEN

ANEXO. EJERCICIOS / EJEMPLOS

cambiar en la BD de tweeter, los campos user.location = “” o null por N/S

Antes del cambio

```
> db.tweets.find({$or : [{"user.location":""}, {"user.location":null}]}).count()
10796
```

Update:

```
> db.tweets.update({$or : [{"user.location":""}, {"user.location": null}]},
        {$set: {"user.location":"N/S"}}, {multi:true})
WriteResult({ "nMatched" : 10796, "nUpserted" : 0, "nModified" : 10796 })
> db.tweets.find({"user.location":"N/S"}).count()
10796
```

actualizar el campo “created_at” que es String a un tipo Date de BSON

```
> var cur = db.tweets.find({},{created_at:1})
> cur.forEach(function(it) {db.tweets.update({_id: it._id},
        {$set:{created_at: new Date(it.created_at)}})})
```

Todos aquellos usuarios que tengan dentro de entidades, menciones de usuarios. Y los 10 usuarios con mas menciones.

```
> db.tweets.count({"entities.user_mentions": {$not:{$size:0}}})
27069

> var cursor = db.tweets.find({"entities.user_mentions":
        {$not:{$size:0}}},{"entities.user_mentions":1})
> cursor.forEach(function(it) {db.tweets.update({_id: it._id},
        {$set:{"mentions_count":it.entities.user_mentions.length}})})
> db.tweets.find({},{"user.screen_name":1,"mentions_count":1}).sort(
        {mentions_count:-1}).limit(10)
```

Tip: hay que tener en cuenta que puede haber mas usuarios que los mostrados con el mismo número de menciones que el mínimo que mostramos. En nuestro caso hay 18 usuarios con 10 menciones.

Do a regular expression query for all users with screen names beginning with ‘a’. Verify that this uses an index

```
> db.tweets.ensureIndex({"user.screen_name": 1})
> db.tweets.find({"user.screen_name": /^a/},
        {"user.screen_name":1, _id: 0}).explain()
```

Using the twitter collection, build a simple index on a user’s friends count. Then build a separate compound index on both friends count and followers counts. Run a few queries against these fields using explain(), and notice which indexes the query optimizer uses

```
> db.tweets.ensureIndex({"user.friends_count":1})
> db.tweets.ensureIndex({"user.friends_count":1, "user.followers_count":1})
```

```
> db.tweets.find({}).sort({"user.friends_count": -1}).explain()
> db.tweets.find({}).sort({"user.followers_count": -1}).explain() //ERROR
> db.tweets.find({}).sort({"user.friends_count":1,
        "user.followers_count": 1}).explain()
```

18.1 Framework de agregación

- By screen name who tweeted the most?

```
> db.tweets.aggregate( {$group: {_id: "$user.screen_name", count: {$sum: 1}}},
        {$sort:{count:-1}},{$limit: 1} )
{ "_id" : "behcolin", "count" : 8 }
```

- Who mentions the most other (unique) tweeters in their tweets? ...
- Ignoring anyone with no friends or no followers:
  1. Who has the highest Followers to Friends (follows) Ratio?

```
> db.tweets.aggregate({$match: {"user.friends_count": {$gt: 0}}},
        {$match: {"user.followers_count": {$gt: 0}}},
        {$project: {"user.screen_name": 1,
            "user.friends_count": 1,
            "user.followers_count": 1,
            _id: 0,
            ratio: {$divide:["$user.followers_count",
                             "$user.friends_count"]}
        }},
        {$sort:{ratio: -1}},
        {$limit: 1})

{ "user" : { "friends_count" : 8, "screen_name" : "Twitterrific",
        "followers_count" : 153772 }, "ratio" : 19221.5 }
```

  2. Who has the lowest?

```
> db.tweets.aggregate({$match: {"user.friends_count": {$gt: 0}}},
        {$match: {"user.followers_count": {$gt: 0}}},
        {$project: {
            "user.screen_name": 1,
            "user.friends_count": 1,
            "user.followers_count": 1,
            _id: 0,
            ratio: {$divide:["$user.followers_count",
                             "$user.friends_count"]}
        }},
        {$sort:{ratio: 1}},
        {$limit: 1})

{ "user" : { "friends_count" : 121, "screen_name" : "whaider2009",
        "followers_count" : 1 }, "ratio" : 0.008264462809917356 }
```

  3. What's the average?

```
> db.tweets.aggregate({$match: {"user.friends_count": {$gt: 0}}},
        {$match: {"user.followers_count": {$gt: 0}}},
        {$project: {
            "user.screen_name": 1,
            "user.friends_count": 1,
            "user.followers_count": 1,
            _id: 0,
            ratio: {$divide:["$user.followers_count",
```

```
                                    "$user.friends_count"]}
                }},
                {$group:{_id: 1, avg: {$avg: "$ratio"}}})

    { "_id" : 1, "avg" : 8.044204413696619 }
```

4. Who is closest to the average? ...

- Sacar de forma ordenada decreciente la diferencia entre followers y friends (mas followers que friends) <- **Asegurar un índice!!**

```
> db.tweets.aggregate({$match: {"user.friends_count": {$gt: 0}}},
        {$match: {"user.followers_count": {$gt: 0}}},
        {$project: {
            "user.screen_name": 1,
            "user.friends_count": 1,
            "user.followers_count": 1,
            _id: 0,
            ratio: {$divide:["$user.followers_count", "$user.friends_count"]},
            diferencia: {$subtract: [
                "$user.followers_count","$user.friends_count"]}
        }},
        {$sort:{ratio: -1}},
        {$limit: 10})

{ "user" : { "friends_count" : 8, "screen_name" : "Twitterrific",
        "followers_count" : 153772 }, "ratio" : 19221.5,
        "diferencia" : 153764 }
{ "user" : { "friends_count" : 2, "screen_name" : "steve_berra",
        "followers_count" : 34248 }, "ratio" : 17124,
        "diferencia" : 34246 }
{ "user" : { "friends_count" : 1, "screen_name" : "2dopeboyz",
        "followers_count" : 16847 }, "ratio" : 16847,
        "diferencia" : 16846 }
{ "user" : { "friends_count" : 9, "screen_name" : "backstreetboys",
        "followers_count" : 125048 }, "ratio" : 13894.222222222223,
        "diferencia" : 125039 }
{ "user" : { "friends_count" : 3, "screen_name" : "tedouumdado",
        "followers_count" : 39467 }, "ratio" : 13155.666666666666,
        "diferencia" : 39464 }
{ "user" : { "friends_count" : 1, "screen_name" : "3in1of2",
        "followers_count" : 10962 }, "ratio" : 10962,
        "diferencia" : 10961 }
{ "user" : { "friends_count" : 1, "screen_name" : "craigslistjobs",
        "followers_count" : 8742 }, "ratio" : 8742,
        "diferencia" : 8741 }
{ "user" : { "friends_count" : 1, "screen_name" : "LaJornada",
        "followers_count" : 8720 }, "ratio" : 8720,
        "diferencia" : 8719 }
{ "user" : { "friends_count" : 6, "screen_name" : "trampos",
        "followers_count" : 35508 }, "ratio" : 5918,
        "diferencia" : 35502 }
{ "user" : { "friends_count" : 1, "screen_name" : "StyleCareers",
        "followers_count" : 3661 }, "ratio" : 3661,
        "diferencia" : 3660 }
```

- Obtener una colección tipo:

```
{_id: hashtag, tweets: [ {id: "", text: "", screen_name: ""}, ...]}
```

Es decir, tag y los tweets relacionados con el tweet

Con push:

```
> db.tweets.aggregate( {$match: {"entities.hashtags": {$not: {$size: 0} } }},
        {$project: {
            _id: 0,
            id:1,
            text: 1,
            "entities.hashtags": 1,
            "user.screen_name":1}},
        {$unwind:"$entities.hashtags"},
        {$group:{
            _id: "$entities.hashtags.text",
            tweets: {$push: {
                        id: "$id",
                        text: "$text",
                        screen_name: "$user.screen_name"}}
        }}, ,
        {$out: "hashtags_tweets"})
```

Con addToSet (si no existe lo crea como array, si existe como array lo añade, **si existe y no es un array da un error**):

```
> db.tweets.aggregate( {$match: {"entities.hashtags": {$not: {$size: 0} } }},
        {$project: {
            _id: 0,
            id:1,
            text: 1,
            "entities.hashtags": 1,
            "user.screen_name":1}},
        {$unwind:"$entities.hashtags"},
        {$group:{
            _id: "$entities.hashtags.text",
            tweets: {$addToSet: {
                        id: "$id",
                        text: "$text",
                        screen_name: "$user.screen_name"}}
        }},
        {$out: "hashtags_tweets"}
    ).pretty()
```

CHAPTER

NINETEEN

ANEXO. EJEMPLOS DE DESPLIEGUES EN LA MISMA MÁQUINA

Ejemplos con máquinas vagrant. Usan el directorio `/home/vagrant/mongodata` para almacenar los ficheros de datos.

Los ejemplos de linea de comandos pueden ser pasados a ficheros de configuración con el antiguo formato.

Para lanzar los demonios usando los ficheros de configuración (YAML o antiguo formato) usar los comandos `-f` ó `--config`.

Estos despliegues crearán todos los demonios en la misma máquina usando distintos puertos

19.1 Linea de comandos

19.1.1 Standalone

```
mkdir -p /home/vagrant/mongodata/standalone/db

mongod --dbpath=/home/vagrant/mongodata/standalone/db --journal \
    --fork --logpath=/home/vagrant/mongodata/standalone/db.log --logappend

mongo
```

19.1.2 Replica Set

```
mkdir -p /home/vagrant/mongodata/replicaSet/{node1,node2,node3}/db

mongod --dbpath=/home/vagrant/mongodata/replicaSet/node1/db --journal \
    --fork --logpath=/home/vagrant/mongodata/replicaSet/node1/db.log \
    --logappend --replSet set0 --port 28001
mongod --dbpath=/home/vagrant/mongodata/replicaSet/node2/db --journal \
    --fork --logpath=/home/vagrant/mongodata/replicaSet/node2/db.log \
    --logappend --replSet set0 --port 28002
mongod --dbpath=/home/vagrant/mongodata/replicaSet/node3/db --journal \
    --fork --logpath=/home/vagrant/mongodata/replicaSet/node3/db.log \
    --logappend --replSet set0 --port 28003

mongo --port 28001
```

Inicialización del replica set

```
config = { "_id": "set0", "members" : [
    {"_id" : 0, "host" : "precise32:28001" },
    {"_id" : 1, "host" : "precise32:28002" },
    {"_id" : 2, "host" : "precise32:28003" } ]
}
rs.initiate(config)
```

19.1.3 Shard

```
mkdir -p /home/vagrant/mongodata/shard/rs{1,2}/node{1,2,3}/db
mkdir -p /home/vagrant/mongodata/shard/config_server_{1,2,3}

mongod --dbpath=/home/vagrant/mongodata/shard/rs1/node1/db --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/rs1/node1/db.log \
    --logappend --replSet set1 --port 28001 --shardsvr
mongod --dbpath=/home/vagrant/mongodata/shard/rs1/node2/db --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/rs1/node2/db.log \
    --logappend --replSet set1 --port 28002 --shardsvr
mongod --dbpath=/home/vagrant/mongodata/shard/rs1/node3/db --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/rs1/node3/db.log \
    --logappend --replSet set1 --port 28003 --shardsvr

mongod --dbpath=/home/vagrant/mongodata/shard/rs2/node1/db --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/rs2/node1/db.log \
    --logappend --replSet set2 --port 28011 --shardsvr
mongod --dbpath=/home/vagrant/mongodata/shard/rs2/node2/db --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/rs2/node2/db.log \
    --logappend --replSet set2 --port 28012 --shardsvr
mongod --dbpath=/home/vagrant/mongodata/shard/rs2/node3/db --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/rs2/node3/db.log \
    --logappend --replSet set2 --port 28013 --shardsvr

mongod --dbpath=/home/vagrant/mongodata/shard/config_server_1 --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/config_server_1/mongod.log \
    --logappend --port 28101 --configsvr
mongod --dbpath=/home/vagrant/mongodata/shard/config_server_2 --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/config_server_2/mongod.log \
    --logappend --port 28102 --configsvr
mongod --dbpath=/home/vagrant/mongodata/shard/config_server_3 --journal \
    --fork --logpath=/home/vagrant/mongodata/shard/config_server_3/mongod.log \
    --logappend --port 28103 --configsvr

mongos --configdb precise32:28101,precise32:28102,precise32:28103 \
    --logpath=/home/vagrant/mongodata/shard/mongos.log --logappend \
    --fork --port 28000
```

Inicialización de los replica set

```
# Conectar antes a un nodo del replica set (mongo --port 28001)
config = { "_id": "set1", "members" : [
    {"_id" : 0, "host" : "precise32:28001" },
    {"_id" : 1, "host" : "precise32:28002" },
    {"_id" : 2, "host" : "precise32:28003" } ]
}
rs.initiate(config)

# Conectar antes a un nodo del replica set (mongo --port 28011)
config = { "_id": "set2", "members" : [
    {"_id" : 0, "host" : "precise32:28011" },
    {"_id" : 1, "host" : "precise32:28012" },
    {"_id" : 2, "host" : "precise32:28013" } ]
}
rs.initiate(config)
```

Inicialización de los shard

```
# Conectar antes a mongos (mongo --port 28000)
sh.addShard("set1/precise32:28001")
sh.addShard("set2/precise32:28011")
```

Activando sharding en una base de datos y tabla

```
sh.enableSharding("mydb")
#Se necesita un índice sobre la shard key si no es _id
db.testtb.ensureIndex({dummy:1})
sh.shardCollection("mydb.testtb", {dummy:1})
```

19.2 YAML

Nuevo formato de configuración. Estos ficheros .yml son los ficheros equivalentes a los ejemplos anteriores.

Sustituyen el lanzamiento del demonio `mongod` para pasarle solo el fichero de configuración:

```
mongod -f configfile.yml
```

19.2.1 Standalone

```
systemLog:
  destination: file
  path: /home/vagrant/mongodata/standalone/db.log
  logAppend: true
processManagement:
  fork: true
storage:
  dbPath: /home/vagrant/mongodata/standalone/db
  journal:
    enabled: true
```

19.2.2 Replica Set

```
# node1_conf.yml file
systemLog:
  destination: file
  path: /home/vagrant/mongodata/replicaSet/node1/db.log
  logAppend: true
processManagement:
  fork: true
net:
  port: 28001
storage:
  dbPath: /home/vagrant/mongodata/replicaSet/node1/db
  journal:
    enabled: true
replication:
  replSetName: set0

# node2_conf.yml file
systemLog:
  destination: file
  path: /home/vagrant/mongodata/replicaSet/node2/db.log
  logAppend: true
processManagement:
  fork: true
net:
  port: 28002
storage:
  dbPath: /home/vagrant/mongodata/replicaSet/node2/db
  journal:
    enabled: true
replication:
```

```
  replSetName: set0

# node3_conf.yml file
systemLog:
  destination: file
  path: /home/vagrant/mongodata/replicaSet/node3/db.log
  logAppend: true
processManagement:
  fork: true
net:
  port: 28003
storage:
  dbPath: /home/vagrant/mongodata/replicaSet/node3/db
  journal:
    enabled: true
replication:
  replSetName: set0
```

19.2.3 Shard

```
#rs1_node1_conf.yml
systemLog:
  destination: file
  path: /home/vagrant/mongodata/shard/rs1/node1/db.log
  logAppend: true
processManagement.fork: true
net.port: 28001
storage:
  dbPath: /home/vagrant/mongodata/shard/rs1/node1/db
  journal.enabled: true
replication.replSetName: set1
sharding.clusterRole: shardsvr

#... (Similar en el resto de nodos del mismo replica set 1)

#rs2_node1_conf.yml
systemLog:
  destination: file
  path: /home/vagrant/mongodata/shard/rs2/node1/db.log
  logAppend: true
processManagement.fork: true
net.port: 28011
storage:
  dbPath: /home/vagrant/mongodata/shard/rs2/node1/db
  preallocDataFiles: false
  smallFiles: true
  journal.enabled: true
replication.replSetName: set2
sharding.clusterRole: shardsvr

#... (Similar en el resto de nodos del mismo replica set 2)

#config_server1_conf.yml
systemLog:
  destination: file
  path: /home/vagrant/mongodata/shard/config_server_1/mongod.log
  logAppend: true
processManagement.fork: true
net.port: 28101
storage:
  dbPath: /home/vagrant/mongodata/shard/config_server_1
```

```
  journal.enabled: true
sharding.clusterRole: configsvr

#... (Similar en el resto de nodos de config servers)

#mongos.yml
systemLog:
  destination: file
  path: /home/vagrant/mongodata/shard/mongos.log
  logAppend: true
processManagement.fork: true
net.port: 28000
sharding.configDB: "precise32:28101,precise32:28102,precise32:28103"
```

CHAPTER

TWENTY

ANEXO. EJEMPLOS DE DESPLIEGUES EN CLUSTER

Ejemplo de despligue de un shard cluster con 2 shards y 3 nodos en cada replica set usando distintas máquinas.

Información de la red:

- 3 máquinas:
  - nodo1: ip: “192.168.50.101”
  - nodo2: ip: “192.168.50.102”
  - nodo3: ip: “192.168.50.103”
- En cada máquina:
  - shard_server set1 port: 27001
  - shard_server set2 port: 27002
  - configserver port: 28000
  - mongos port: 27017 (default port)

Montaje de los replica set:

Nos conectamos a un nodo de cada uno de los replica set e inicializamos los replica set.

```
# Replica set 1 (mongo --host 192.168.50.101:27001)
config = { "_id": "set1", "members" : [
    {"_id" : 0, "host" : "192.168.50.101:27001" },
    {"_id" : 1, "host" : "192.168.50.102:27001" },
    {"_id" : 2, "host" : "192.168.50.103:27001" } ]
}
rs.initiate(config)

# Replica set 2 (mongo --host 192.168.50.101:27002)
config = { "_id": "set2", "members" : [
    {"_id" : 0, "host" : "192.168.50.101:27002" },
    {"_id" : 1, "host" : "192.168.50.102:27002" },
    {"_id" : 2, "host" : "192.168.50.103:27002" } ]
}
rs.initiate(config)
```

Nos conectamos a mongos e inicializamos el shard.

```
sh.addShard("set1/192.168.50.101:27001")
sh.addShard("set2/192.168.50.101:27002")
```

Plantillas de aprovisionamiento:

```
# mongo db shardsvr config
systemLog:
  destination: file
  path: /home/vagrant/mongodata/{{set_name}}/db.log
  logAppend: true
```

```
processManagement:
  fork: true
net:
  port: {{port_number}}
storage:
  dbPath: /home/vagrant/mongodata/{{set_name}}
  journal:
    enabled: true
replication:
  replSetName: {{set_name}}
sharding:
  clusterRole: shardsvr

# mongo db configsvr
systemLog:
  destination: file
  path: /home/vagrant/mongodata/configsvr/db.log
  logAppend: true
processManagement:
  fork: true
net:
  port: 28000
storage:
  dbPath: /home/vagrant/mongodata/configsvr
  journal:
    enabled: true
sharding:
  clusterRole: configsvr

# mongos
systemLog:
  destination: file
  path: /home/vagrant/mongodata/mongos/db.log
  logAppend: true
processManagement:
  fork: true
net:
  port: 27017
sharding:
  configDB: {{shard_configDB}}
```